UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
MESTRADO EM ENGENHARIA ELÉTRICA

USO DE ALGORITMOS MEMÉTICOS NA OTIMIZAÇÃO DE SEQUÊNCIAS DE MONTAGEM DE MÁQUINAS SMD

JOSÉ ELIDELSON DA COSTA CARVALHO

MANAUS
2007

UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
MESTRADO EM ENGENHARIA ELÉTRICA

JOSÉ ELIDELSON DA COSTA CARVALHO

USO DE ALGORITMOS MEMÉTICOS NA OTIMIZAÇÃO DE SEQUÊNCIAS DE MONTAGEM DE MÁQUINAS SMD

Dissertação apresentada ao Programa de Mestrado em Engenharia Elétrica da Faculdade de Tecnologia da Universidade Federal do Amazonas, como requisito parcial para a obtenção do título de Mestre em Engenharia Elétrica, área de concentração Controle e Automação de Sistemas.

Orientador: Prof Dr João Edgar Chaves Filho

MANAUS
2007

JOSÉ ELIDELSON DA COSTA CARVALHO

USO DE ALGORITMOS MEMÉTICOS NA OTIMIZAÇÃO DE SEQUÊNCIAS DE MONTAGEM DE MÁQUINAS SMD

Dissertação apresentada ao Programa de Mestrado em Engenharia Elétrica da Faculdade de Tecnologia da Universidade Federal do Amazonas, como requisito parcial para a obtenção do título de Mestre em Engenharia Elétrica, área de concentração Controle e Automação de Sistemas**.**

Aprovado em 28 de dezembro de 2007.

BANCA EXAMINADORA

Prof Dr João Edgar Chaves Filho, Presidente
Universidade Federal do Amazonas

Prof. Dr. Cícero Ferreira F. Costa Filho, Membro
Universidade Federal do Amazonas

Prof. Dr. Raimundo Barreto, Membro
Universidade Federal do Amazonas

Agradeço a minha esposa Eliane,
ao meu filho Judah pelo incentivo
e motivação para a realização
deste trabalho.

Agradecimentos,

A Deus pela Sabedoria e Inteligência;

A minha esposa e meu filho pelo apoio e motivação;

Ao meu orientador pelo acompanhamento e pelo apoio;

A SONY Brasil por permitir os testes reais no seu setor de Inserção/Automática/SMD;

Ao Centro de Tecnologia Eletrônica e da Informação – CETELI – UFAM e a SUFRAMA, através dos convênios n.068 e 069 de 2001.

AGRADEÇO

RESUMO

A otimização da seqüência de montagem de componentes SMD em placas de circuito impresso tem sido alvo de intensa pesquisa por ser um dos pontos fundamentais para a eficiência de linhas de produção de placas em indústrias de produtos eletrônicos. Entre as diversas técnicas utilizadas para resolver este tipo de problema estão os Algoritmos Genéticos chamados também de evolucionários por sua analogia com a evolução biológica natural das espécies. Outro tipo de algoritmo evolucionário chamado de Algoritmo Memético tem apresentado melhores resultados que os Algoritmos Genéticos em diversas áreas de pesquisa. Portanto este trabalho propõe uma investigação do uso deste algoritmo na resolução do problema da otimização da seqüência de montagem de componentes SMD. Foram feitos diversos testes usando Algoritmos Genéticos e Meméticos em diferentes seqüências de montagem e os resultados mostraram um melhor desempenho dos Algoritmos Meméticos em relação aos Genéticos. Portanto os Algoritmos Meméticos se mostraram uma promissora ferramenta para a otimização deste problema.

Palavras chave: Otimização, SMD

ABSTRACT

The optimization of SMD electronics components assembly in printed circuit boards has been target of intensive research for it is one of the important point in production lines efficiency of the electronic industry. Among many techniques utilized for solving this kind of problem are the Genetics Algorithms also called Evolutionary Algorithms due it analogy with natural biologic evolution. Another kind of Evolutionary Algorithms called Memetics Algorithm has presented better results than Genetics Algorithms in many application fields. So this work proposes an investigation about using this algorithm for solving the SMD sequence placement problem. Many tests were done using Genetic and Memetics Algorithms on different placement sequence sets and the results showed a better performance of Memetic Algorithms related to Genetic Algorithms. So Memetics Algorithms has been showed to be an important tool on solving the problem of SMD placement sequence.

Key words: Optmization, SMD

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AG – Algoritmo Genético

AM – Algoritmo Memético

ANC – *Automatic Nozzle Change* (Troca Automática de Nozzle)

BT – Busca Tabu (método de busca local)

Ganho % – Ganho (redução) percentual no tempo de montagem

Hill-Climbing – Algoritmo Escalada de Montahnha

Multiple Start – Algoritmo Multiplos Inícios

PAPSP – *Pick and Place Sequencing Problem* (Problema de Roteamento de Componentes)

PCI – Placa de Circuito Impresso

RAP – *Rheel Assignment Problem* (Problema de Configuração dos carretéis nos slots)

SA – *Simulated Annealing* (método de busca local)

Slot – Local para colocação de carretel de componentes

SMD – *Surface Mounted Device* (Dispositivo de Montagm em Superfície)

SMT – *Surface Mounting Technology* (Tecnologia de Montagem em Superficie)

SMM – *Surface Mounting Machine* (Máquina de Montagem em Superfície)

TAP – *Tool Assigmnent Problem* (Problema de Configuração dos Nozzles)

TEA – Tempo de Execução dos Algoritmos

TMP – Tempo de Montagem das Placas

TSP – *Traveling Salesman Problem* (Problema do Caixeiro Viajante)

VNS – *Variable neighborhood*

Capítulo 01

Introdução

1.1 A montagem de placas de circuito impresso e seus problemas associados

A demanda por produtos sofisticados nas últimas décadas na indústria eletrônica fez crescer a necessidade de placas de circuito impresso com quantidades e tipos cada vez maiores de componentes.

Como uma solução para este problema surgiu a tecnologia chamada SMD a qual utiliza componentes eletrônicos de pequenas dimensões em substituição aos chamados componentes convencionais ou *through-hole.* Esta tecnologia possibilitou então o projeto de PCI's com grande quantidade e diversidade de componentes.

Por sua vez as linhas de montagem de PCI's apresentam diversos níveis de problemas passíveis de otimização e que estão relacionados entre si, os quais estão descritos a seguir:

- **Problema de PCP (planejamento e controle de produção):**

  Neste contexto o problema consiste em como realizar os programas semanais e mensais de produção;

- **Problema de balanceamento de linha:**

  Neste caso o problema consiste em como dividir a carga de trabalho entre as máquinas para a produção de um determinado modelo;

- **Problema de otimização de tempo de preparação de máquinas:**

  Aqui o problema consiste em minimizar o tempo de preparação de máquina quando há a troca de modelos. A idéia e encontrar um arranjo comum de alimentadores que atenda ao maior número de modelos possíveis.

- **Problema de otimização de uma única máquina:**

  Este problema é onde se concentra o objetivo deste trabalho e geralmente é subdivido nos seguintes subproblemas normalmente conhecidos por suas siglas em inglês PAPSP, RAP e TAP:

  - Otimização da seqüência de montagem (PAPSP);
  - Otimização da alocação de carretéis de componentes (RAP);
  - Otimização da alocação de ventosas nos cabeçotes (TAP);

Os subproblemas PAPSP e RAP são de natureza combinatória e conhecidos por sua complexidade computacional, o que significa longos e geralmente inaceitáveis tempos de processamento. Apesar de serem fortemente interdependentes, são considerados difíceis de serem resolvidos concorrentemente.

Nos últimos anos os Algoritmos Evolucionários, chamados assim por se basearem na teoria da evolução biológica das espécies, tem sido bastante utilizados para resolver os problemas de otimização de máquina descritos acima, com destaque para a utilização dos Algoritmos Genéticos. Existe, entretanto, um tipo de Algoritmo Evolucionário chamado de Algoritmo Memético que tem apresentado resultados ainda melhores que os Algoritmos Genéticos em diversas áreas de aplicação, fornecendo soluções mais refinadas ou mais próximas da solução ótima. Assim, fizemos uma revisão bibliográfica para averiguar como os Algoritmos Meméticos foram utilizados na solução de problemas de otimização.

1.2 Justificativa da pesquisa

A falta de referências na pesquisa bibliográfica analisada sobre o uso dos Algoritmos Meméticos na otimização da montagem de PCI's e os seus significativos resultados de desempenho superior aos dos Algoritmos Genéticos em diversas áreas de aplicação, foram a principal motivação e justificativa para este trabalho.

1.3 Objetivos

1.3.1 Geral

Investigar o desempenho de Algoritmos Meméticos na otimização do tempo de seqüência de montagem de componentes SMD em placas de circuito impresso.

1.3.2 Específicos

- Propor uma solução alternativa ou mais refinada em relação às obtidas com os diversos trabalhos publicados sobre o problema da otimização do tempo de montagem de componentes SMD;
- Comparar o desempenho do Algoritmo Memético com o Algoritmo Genético neste tipo de problema de otimização.

1.4 Organização do Trabalho

Este trabalho está organizado em oito capítulos mais um apêndice.

O Capítulo 1 apresenta uma introdução, os objetivos do trabalho e a organização do texto.

No Capítulo 2 encontra-se a revisão bibliográfica.

No Capítulo 3 é feita uma descrição do problema a ser resolvido.

O Capítulo 4 apresenta uma formulação matemática do problema.

A descrição metodológica é abordada no capítulo 5.

O capítulo 6 apresenta os resultados obtidos e sua análise.

As conclusões e pesquisas futuras são mostradas no capítulo 7.

Capítulo 02

Revisão Bibliográfica

2.1 Algoritmos Evolucionários

Os algoritmos evolucionários, baseados no seu uso com sucesso em diversas áreas que necessitam de métodos de otimização, surgem como uma alternativa para a solução dos problemas associados à montagem de PCI´s tanto de forma isolada como concorrente. Neste trabalho vamos nos concentrar em particular nos Algoritmos Meméticos e Genéticos.

As principais vantagens dos algoritmos Meméticos e Genéticos são listadas a seguir:

- São robustos para problemas altamente não lineares;
- Normalmente conseguem evitar a queda em mínimos locais do espaço de soluções.

Este trabalho usa o método baseado em AG proposto por Lee et al [2], o qual está descrito no capítulo 5, para comparar os seus resultados com os resultados obtidos pelo uso de Algoritmos Meméticos. Esse método apresenta excelentes resultados quando comparado a métodos comumente utilizados na indústria para otimização do RAP e PAPSP.

Embora os Algoritmos Genéticos tenham apresentado bons resultados, eles geralmente apresentam soluções sub-ótimas. Na tentativa de obter melhores resultados os pesquisadores têm usado, com sucesso, os Algoritmos Mémeticos (AM).

A vantagem dos AM's sobre os AG's é que eles conseguem soluções mais refinadas (vide as referências bibliográficas), ou seja, com melhores valores de Adaptação. Isto é conseguido, por que os AM's usam métodos de busca local, em adição aos métodos de Cruzamento e Mutação dos AG's.

No apêndice, os algoritmos Meméticos e Genéticos são mostrados com mais detalhes.

2.2 Algoritmos Genéticos

Lee et al [2] desenvolveram uma solução onde utilizam AG para otimizar o arranjo de ventosas nos cabeçotes (TAP), a carga de trabalho entre os cabeçotes, e por fim otimizar simultaneamente o PAPSP e o RAP utilizando *links* (cromossomos) parciais para a alocação de carretéis de componentes e a sua seqüência de posicionamento. Esse método pode ser aplicado tanto para máquinas *Single Head* quanto para máquinas *Multi Head*. Leu et al [22] também desenvolveram uma solução para otimizar simultaneamente o PAPSP e RAP para máquinas SMM do tipo *turret* . Neste arranjo eles consideraram também dois *links*, sendo o primeiro link para a seqüência de posicionamento dos componentes e o segundo para o arranjo dos carretéis. Eles aplicaram quatro operadores genéticos em cada *link* e a função de Adaptação era o tempo total de montagem da placa. Wang et al [23] usaram AG para otimizar o arranjo de carretéis para máquinas Fuji QP-122. Khoo and Loh [6] também usaram AG para otimizar simultaneamente o PAPSP e o RAP. Eles afirmam ter obtido resultados melhores do que Leu et all.

2.3 Algoritmos Meméticos

Shawki et al [14] fizeram um estudo comparativo e mostraram o melhor desempenho dos AM´s em relação aos AG´s e outros métodos heurísticos tais como: Métodos Interativos e Aproximação por autovetores matriciais. Os resultados desta pesquisa mostraram que a maioria das soluções obtidas pelos Algoritmos Meméticos e GRASP foram superiores às soluções obtidas com Algoritmos Genéticos e Aproximação por autovetores. No trabalho foi abordado o problema de particionamento de circuitos.

Peter Merz e Bernd Freisleben [15] apresentam um interessante trabalho sobre melhoria da eficiência de métodos de busca local para Algoritmos Genéticos, o que é a base dos

Algoritmos Meméticos. Eles concluíram que estruturas de dados bem elaboradas em conjunto com o método *Hill-Climbing* de busca local são pontos fundamentais nos desempenhos de tempo.

Natalio Krasnogor [16] escreveu um excelente tutorial sobre Algoritmos Meméticos com conceitos e definições importantes sobre esta técnica de otimização. Além disso, apresentou diversos exemplos de possíveis implementações de Algoritmos Meméticos que servem de base para implementações reais. Marcone Jamilson [25] apresentou um excelente tutorial sobre heurísticas de busca local, incluindo as heurísticas Busca Tabu e *Simulated Annealing* as quais são usadas neste trabalho. Ender Ozcan e Murat Erenturk [17] apresentaram um ótimo trabalho sobre a resolução do problema do caixeiro viajante usando Algoritmos Meméticos. O trabalho é interessante, pois os autores abordaram diversos tipos de operadores genéticos de crossover e mutação. Marcos Goldbarg et al [18] usaram com sucesso os AM´s na determinação da configuração de custo mínimo de sistemas de co-geração de energia com base no gás natural. O problema foi classificado como computacionalmente difícil devido a restrições técnicas, operacionais e econômicas. Algoritmos Meméticos e Genéticos foram aplicados em um conjunto de 35 instâncias geradas de acordo com cinco diferentes ciclos de co-geração. O trabalho concluiu que os Algoritmos Meméticos suplantam, de uma forma geral a performance dos Algoritmos Genéticos ressaltando a importância da busca local.

Alexandre Mendes [13] mostrou o poder dos Algoritmos Meméticos aplicando-os no problema de sequenciamento de máquinas. Ele usou o Algoritmo *Multiple Start* como comparativo de desempenho em relação aos Algoritmos Meméticos. Como conclusão interessante do trabalho, foi verificado que o uso de populações hierarquicamente estruturadas, a redução da vizinhança para a busca local e o ajuste automático dos parâmetros do Algoritmo Memético foram fatores importantes para o seu melhor desempenho.

Ho e Ji [24] usaram o algoritmo conhecido como *Nearest Neighbour* como método de busca local na solução dos problemas de PAPSP e RAP para uma máquina SMM do tipo *Chip Shooter*.

2.4 Outros Métodos

Ball e Magazine [19] formularam o PAPSP como o clássico problema do caixeiro viajante (TSP). Leipälä e Nevalainen [20] trataram O PAPSP como um TSP assimétrico e tridimensional, enquanto o RAP foi modelado como um problema de alocação quadrático. Kumar e Li [21] formularam o RAP como um problema de acoplamento de peso mínimo (*Minimum Weight Matching Problem*) e o PAPSP como um TSP. Burke et.al [4] definiram um modelo onde consideram o PAPSP e o RAP simultaneamente e usaram o método chamado VNS (*variable neighborhood*).

Capítulo 03

Descrição do problema

3.1 A tecnologia SMD e máquinas SMM

Enquanto os componentes *through-hole* são inseridos nas PCI's através de seus terminais por máquinas chamadas de insersoras, os componentes SMD levaram à criação das chamadas máquinas SMM, as quais, ao invés de inserir, posicionam os componentes na placa. Estes são fixados nas placas através da aplicação de adesivo ou pasta de solda e posteriormente são soldados. Esses métodos de montagem de componentes SMD são descritos de forma sucinta a seguir:

- **Montagem com adesivo*:***
    - É aplicado adesivo nas posições de montagem dos componentes SMD;
    - Os componentes SMD são posicionados pelas máquinas SMM;
    - A PCI passa em um forno com perfil térmico para que seja feita a cura do adesivo;
    - A PCI passa por uma máquina de solda do tipo onda onde os componentes SMD e *through-hole* (caso estejam presentes) são soldados.
- **Montagem com Pasta de Solda**:
    - É aplicada pasta de solda nas ilhas onde serão posicionados os componentes SMD;
    - Os componentes SMD são posicionados pelas máquinas SMM;
    - A PCI passa em um forno com perfil térmico (forno de refusão) para que seja feita a fusão dos terminais dos componentes com a pasta de solda.

A figura 3.1 abaixo mostra um esquema genérico de uma SMM do tipo *multi head*, cujas partes principais são descritas a seguir:

Vale ressaltar que o uso de termos em inglês é conveniente neste tipo de problema, visto que, se trata de linguagem técnica bastante usada pelos profissionais da área de montagem de PCI's.

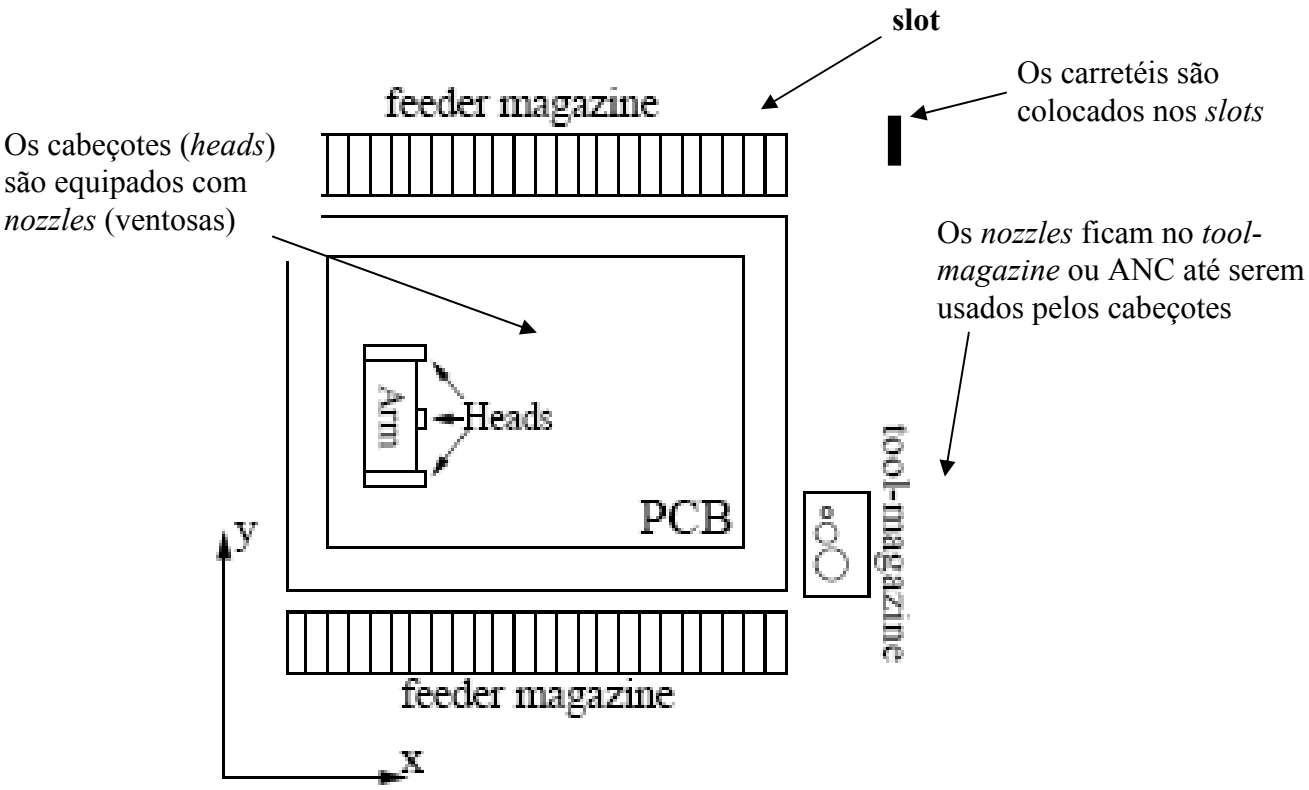


Fig 3.1 – Esquema genérico de uma SMM

- ***Feeder magazine*:**

  Os *feeders* (alimentadores) possuem um determinado número de posições adjacentes chamadas de *slots*. Nos slots são colocados os carretéis (*reels*) com os diversos tipos de componentes SMD.

- ***Heads*:**

As SMM possuem um (*Single Head*) ou mais cabeçotes (*Multi Head*). A função dos cabeçotes é capturar (*pick*) um componente SMD em um determinado *slot* e transportá-lo até a posição onde será montado (*place*);

- ***Tool-magazine* (ANC):**

Para capturar um componente SMD em um *slot* um cabeçote utiliza uma ventosa de ar-comprimido, a qual é chamada de *nozzle. C*omo existem componentes SMD de diversas formas e tamanhos, uma SMM geralmente possui vários tipos de *nozzles*. Durante o processo de posicionamento estes *nozzles*, se for necessário, são trocados no chamado ANC (*automatic nozzle change*).

- ***Arm*:**

O Arm (braço) é onde os cabeçotes são fixados. Ele se movimenta nas direções XY.

- **PCI's:**

As PCI's são transportadas por esteiras (*conveyors*) e param em posições específicas na mesa das máquinas para serem montadas.

A figura 3.2 abaixo mostra os principais tipos de máquinas SMM encontrados na indústria, sendo que o tipo *Pick & Place* Multi Head é o mais usado atualmente.

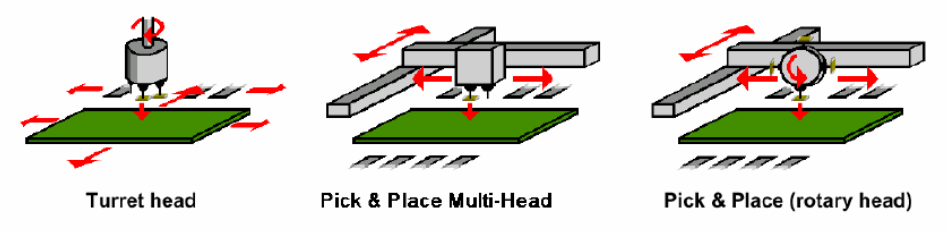


Fig 3.2 – Principais tipos de máquinas SMM

As máquinas *Turret head* e *Rotary head* combinam movimentos de translação do braço com rotação da cabeça enquanto a *multi-head* usa somente movimentos de translação.

3.2 Problemas associados à montagem de PCI's

A montagem de PCI's em máquinas SMM é feita baseada em programas que são armazenados ou carregados na memória das máquinas e executados. Para cada modelo de placa existe um programa associado. Estes programas são criados normalmente no formato texto e tem como principal objetivo, determinar a seqüência de montagem dos componentes e o arranjo ou alocação desses componentes nos *slots* da máquina.

Neste trabalho serão consideradas máquinas SMM do tipo *Pick & Place*, sendo que os métodos e resultados aqui apresentados podem ser usados tanto para máquinas *Single Head* quanto para máquinas *Multi Head*.

Durante a execução dos programas citados acima os seguintes passos são executados pelas máquinas SMM:

- Os cabeçotes são equipados com os *nozzles* apropriados para o primeiro grupo de componentes a ser posicionado;
- Os braços (*arms*), nos quais os cabeçotes estão fixados, se dirigem aos *slots* onde estão estes componentes e cada cabeçote faz a captura (*pick*) do componente correspondente;
- Os braços se deslocam até a posição XY na PCI onde será posicionado o primeiro componente, faz o posicionamento (*place*) e repete a operação para os outros componentes do grupo, onde cada componente está associado a um cabeçote;
- Após o posicionamento do último componente do grupo, o programa verifica se algum componente do próximo grupo precisará de um *nozzle* diferente do que está no cabeçote correspondente. Portanto umas das seguintes situações podem ocorrer:
- O braço se desloca até o ANC (*automatic nozzle change*) e efetua a troca do(s) *nozzle*(s);
- O braço segue diretamente para o próximo grupo de *slots* para fazer a captura dos componentes.
- O ciclo se repete até que o último grupo de componentes seja posicionado.

O objetivo principal de um programa como o descrito acima é otimizar o tempo de montagem da PCI, ou seja, executar a seqüência de montagem no menor tempo possível. Cada segundo reduzido representa ganhos significativos de produção. Programas mal elaborados podem causar os chamados “gargalos de produção”, ou seja, atrasos na linha SMD e conseqüentemente em todo o processo produtivo.

Já foi mostrado no capítulo 1 que o problema da otimização deste tempo de montagem geralmente é particionado em três principais subproblemas, os quais são descritos a seguir:

- **Problema do sequenciamento de componentes (*pick & place sequencing problem*) PAPSP**

  Este problema consiste na determinação da seqüência ótima (menor tempo possível) dos ciclos de montagem de componentes (*pick & place*).

- **Problema da alocação dos carretéis de componentes nos *feeder*s (*reel assignment problem*) RAP**

  Este problema consiste em determinar a alocação ou arrumação ótima dos carretéis de componentes nos *slots*. A alocação dos carretéis influi diretamente no tempo da seqüência de montagem dos componentes descrita acima.

- **Problema da alocação dos nozzles (*tool assignment problem*) TAP**

  Este problema consiste em determinar a alocação ótima dos *nozzles* por cabeçote de forma a minimizar a necessidade da troca de *nozzles* durante os ciclos de *pick* & *place*. A troca de *nozzles* é uma operação extremamente lenta em comparação ao tempo de *pick* de componentes e, portanto, deve ser minimizada.

Capítulo 04

Formulação Matemática

4.1 Introdução

Burke et.al [4] apresentam um bom modelo matemático para representar o problema da otimização do tempo de montagem de PCB`s por máquinas SMM. Neste modelo Burke et al [4] consideram o problema como o clássico problema do caixeiro viajante (TSP), onde as cidades são as posições XY de posicionamento dos componentes SMD e as distâncias entre estas posições (cidades) são calculadas como distâncias Euclidianas. Este modelo está descrito a seguir:

4.2. Modelo para o PAPSP:

- Seja $\{c_1, c_2, \ldots, c_n\}$ o conjunto finito das n posições XY de componentes na PCB;
- D_{ij} será então a distância Euclidiana entre duas posições (componentes) c_i e c_j;
- Seja $\sigma : \{1, 2, .., n\} \rightarrow \{1, 2, .., n\}$ uma permutação das possíveis seqüências de montagem desses componentes.

Então para resolver o PAPSP devemos obter uma seqüência **σ** de forma que tenhamos:

$$\min_{\sigma} \quad \sum_{i=1}^{n-1} d_{\sigma(i),\sigma(i+1)} + d_{\sigma(n),\sigma(1)} \qquad (1)$$

obs: o segundo termo $d_{\sigma(n),\sigma(1)}$ na equação (1) acima e nas próximas equações significa que o processo de montagem é cíclico, ou seja, após a montagem do último componente (n) os cabeçotes voltam à posição inicial 1.

4.3 Modelo para o PAPSP + RAP:

O tempo gasto entre dois posicionamentos executados por uma máquina SMM é obviamente influenciado pela posição dos *slots* alocados para estes componentes. Conseqüentemente o tempo total é influenciado pelo arranjo global dos componentes nos *slots* da máquina. Portanto pode-se modelar esta influência da seguinte forma:

- Seja $\{s_1, s_2, \ldots, s_f\}$ o conjunto finito dos **f** *slots* disponíveis na máquina;
- Seja $\delta : \{c_1, c_2, ..., c_n\} \rightarrow \{s_1, s_2, \ldots, s_f\}$ um mapeamento que associa cada posição XY de componente na PCB a um *slot*;
- Seja $d^{\delta}_{i,j}$ a distancia (ou o tempo gasto) entre o término do posicionamento do componente c_i e o término do posicionamento do componente c_j para um dado arranjo δ de alocação de componentes nos *slots*;

Então para resolver o PAPSP +RAP temos um problema de otimização em dois níveis onde devemos encontrar um $\boldsymbol{\sigma}$ e $\boldsymbol{\delta}$ de forma que tenhamos:

$$\underset{\sigma,\delta}{\text{minimizar}} \quad \sum_{i=1}^{n-1} d^{\delta}_{\sigma(i),\sigma(i+1)} + d^{\delta}_{\sigma(n),\sigma(1)} \qquad (2)$$

4.4 Modelo para o PAPSP + RAP + TAP:

Devido ao fato dos componentes SMD terem formas diferentes, geralmente torna-se necessária a troca de *nozzles* durante o processo de montagem. Esta operação é bem mais demorada do que a operação de captura de componentes (*pick*) e deve ser minimizada.O modelo então considera esta operação da seguinte forma:

- Seja $\{t_1, t_2, \ldots, t_m\}$ o conjunto finito dos m *nozzles* (*tools*) disponíveis na máquina;
- Seja $\tau : \{c_1, c_2, ..., c_n\} \rightarrow \{t_1, t_2, \ldots, t_m\}$ um mapeamento (função) que associa cada posição XY de componente na PCB com um *nozzle* adequado;
- Seja $d_{i,j}^{\delta,\tau}$ o tempo para o cabeçote começar com o *nozzle* $\tau(c_i)$ na posição c_i, fazer a troca de *nozzle* se $\tau(c_i) \neq \tau(c_j)$, capturar o componente no *slot* $\delta(c_j)$ e montá-lo na posição c_j.

Então para resolver PAPSP +RAP+TAP temos um problema de otimização em três níveis onde devemos encontrar um **σ, δ** e **τ** de forma que tenhamos:

$$\underset{\sigma,\ \varrho\ \tau}{\text{minimize}} \sum_{i=1}^{n-1} d_{\sigma(i),\sigma(i+1)}^{\varrho,\tau} + d_{\sigma(n),\sigma(1)}^{\varrho,\tau} \qquad (3)$$

Como uma mudança de *nozzle* consome muito tempo devemos ter que:

$$d_{i,j}^{\delta,\tau} << d_{k,l}^{\delta,\tau} \text{ se } \tau(c_i) = \tau(c_j) \text{ e } \tau(c_k) \neq \tau(c_l).$$

Capítulo 05

Descrição Metodológica

Neste capítulo descrevemos o método de referência usado no trabalho, o Algoritmo Memético e os métodos de busca local utilizados.

5.1 Método de Referência

Nesta seção será descrito o método proposto por Lee at al [2], o qual usa Algoritmos Genéticos na solução dos subproblemas PAPSP, RAP e TAP. Este método mostrou-se bastante eficiente segundo os resultados apresentados no trabalho, além disso, os detalhes apresentados no paper de Lee at al [2] possibilitaram a sua implementação através do MatLab.

Será usado então este algoritmo como base para implementação de um Algoritmo Memético, ou seja, introduziremos métodos de busca local em posições estratégicas do código como será mostrado na próxima seção. O objetivo é comparar o desempenho do AG e AM. Qualquer ganho de tempo conseguido é importante neste tipo de sistema, pois representará aumento de produção e o conseqüentemente aumento da eficiência do processo produtivo.

A seguir apresentaremos os principais pontos do método de Lee at al [2]:

5.2 Definições do método

- Seja N={1, 2, ..., n} um conjunto de n inteiros positivos e C={c_1, c_2, ..., c_n} um conjunto de n genes. Então cada elemento c_i é um gene e $c_i \neq c_j$ se $i \neq j$.
- Um link é definido como $\tilde{l}$ = [l(1), l(2), ..l(k)., l(n)] onde l(k) é o gene na k-ésima posição do link e l é uma função (mapeamento) $l : N \rightarrow C$, e $\tilde{l}(i) \neq \tilde{l}(j)$ se $i \neq j$.

- O conjunto formado L={$\tilde{l}_1$, $\tilde{l}_{2,}$ ..., $\tilde{l}_{w}$} é formado por todos os possíveis links, onde w representa o número de todas as possíveis permutações de genes que formam os links.
- A função de Adaptação é um mapeamento definido como F : L → R, onde R é o conjunto dos números reais.

5.3 Regras do método

O método estabelece as seguintes regras para sua implementação:

- O número de cabeçotes não é limitado;
- A ordem de operação dos cabeçotes será da seguinte forma: o primeiro cabeçote posiciona o seu componente, em seguida o segundo cabeçote faz o posicionamento e assim sucessivamente até o último cabeçote;
- O número de carretéis de componentes utilizados será, no máximo, igual ao número de *slots* disponíveis.

5.4 Operadores Genéticos de Link parcial

O método usa dois *links* (A e B), sendo um para o RAP e o outro para o PAPSP. Os operadores genéticos, então, são aplicados em cada *link*, de forma a resolver conjuntamente o problema da alocação do slots e da seqüência de posicionamento dos componentes.

5.4.1 Operador de *Crossover*

Existem vários operadores de cruzamento (*crossover*) que simulam as combinações genéticas entre os cromossomos de um indivíduo, sempre procurando formar indivíduos com melhor aptidão ou Adaptação. No problema que estamos estudando isto significa obter

seqüências de alocação de *slots* e de posicionamento de componentes que em conjunto apresentem tempos de execução cada vez menores. O operador usado neste método é descrito a seguir:

- Selecionam-se duas posições aleatórias p1 e p2 do cromossomo pai1 ou pai2;
- Coloca-se a seqüência de genes selecionada no cromossomo filho;
- Os genes que faltam no filho são completados a partir dos genes que não foram selecionados de pai2 ou pai1, selecionados da esquerda para a direita, tomando-se o cuidado de verificar se não existem repetições.

A figura 5.1 mostra um exemplo de aplicação deste operador.

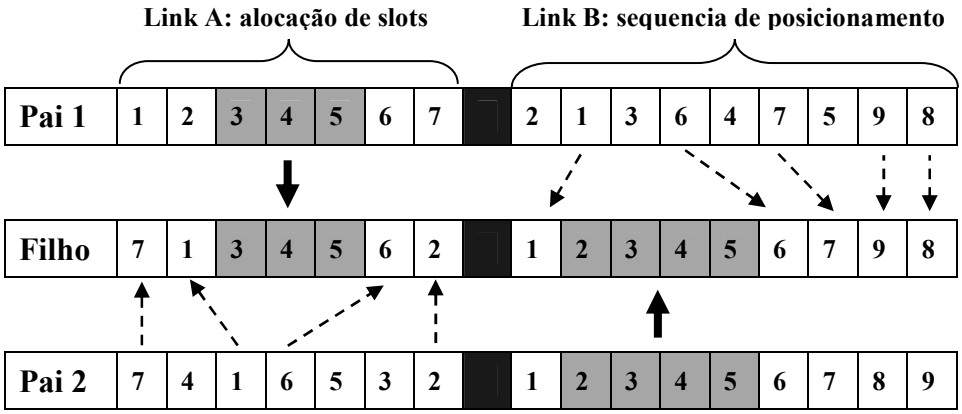


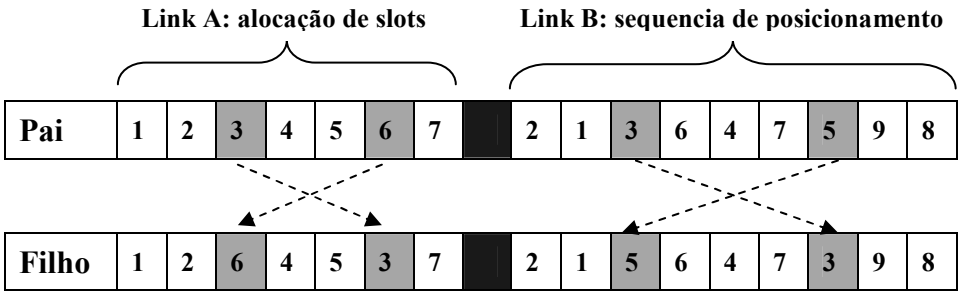
Fig 5.1. Exemplo do Operador Genético *Crossover* com *Links* parciais

5.4.2 Operador de mutação

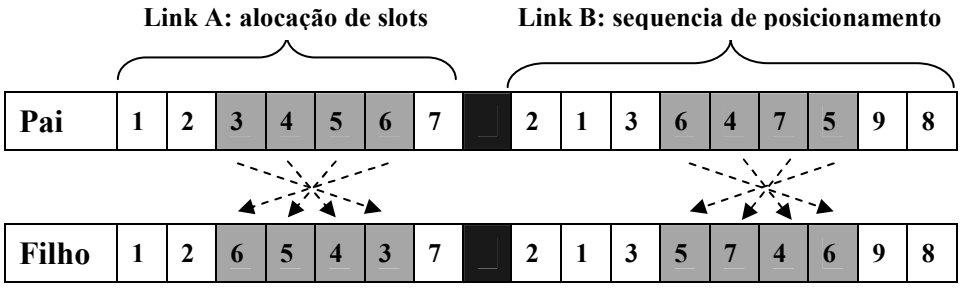
Os operadores de mutação têm uma função especial de evitar que o algoritmo caia em mínimos locais gerando resultados inadequados. Eles conseguem isso por que as operações aplicadas sobre os genes dos cromossomos (soluções) permitem que todo o espaço de busca seja

alcançado, ou seja, qualquer indivíduo, da população possui uma probabilidade de ser selecionado.

A figura 5.2 mostra os operadores de mutação usados neste método. Observa-se que no caso da mutação é necessário apenas um cromossomo pai, o qual, gerará um cromossomo filho após sofrer mutação.



Mutação tipo troca



Mutação tipo inversão

Fig 5.2. Exemplo dos Operadores Genéticos de Mutação com *Links* parciais

5.4.3 Função de Adaptação

A função de Adaptação ou aptidão fornece uma medida do potencial de um individuo da população ou do espaço de busca como solução para o problema de otimização em questão. No

caso da otimização da seqüência de montagem de PCB's por máquinas SMM, um indivíduo é na verdade a junção do arranjo dos *slots* mais a seqüência de posicionamento dos componentes. Por sua vez a função de Adaptação calculará o tempo em que um arranjo de *slots*+seqüência de posicionamento gastará pra montar uma PCB. Portanto a função de Adaptação fará uma simulação do funcionamento da máquina. A implementação desta função dependerá, portanto do tipo de máquina que se está usando.

De uma forma geral a função de Adaptação deverá calcular o tempo ou deslocamento baseado no fluxograma mostrado na figura 5.3:

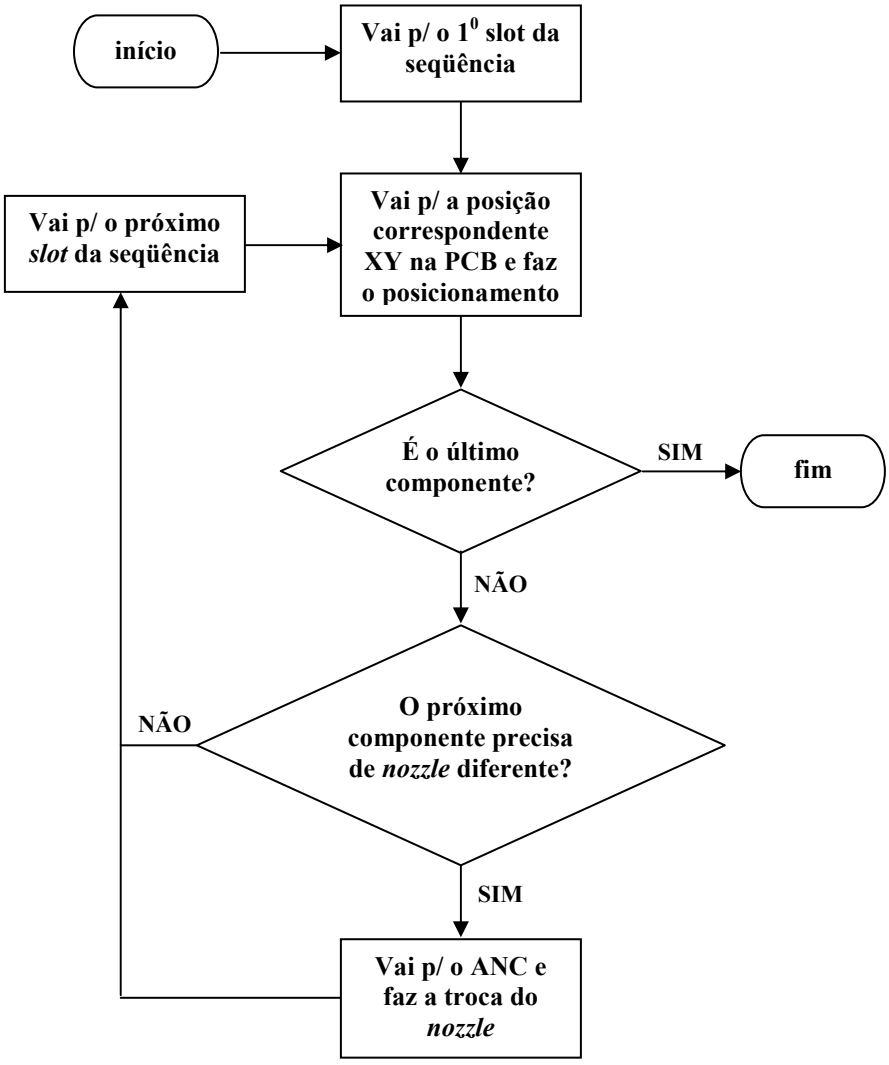


Fig 5.3. Fluxograma básico de funcionamento de uma máquina SMM

5.5 Aplicação para máquinas Single Head

A aplicação do método de referência consiste na criação de dois *links* A e B. Estes *links* serão então processados de forma conjunta pelo AG e AM, através dos operadores genéticos e função de Adaptação, para obter uma solução otimizada.

O link A representa a alocação dos carretéis de componentes nos *slots* da máquina e é definido da seguinte forma:

$l_A = [l_A(1), l_A(2),..., l_A(m)]$ onde $l_A(i) \in S$, $l_A(i) \neq l_A(j)$ e $i \neq j$, $A_{i,j} \in \{1, 2, .., m\}$, onde:

gene $l_A(i)$: representa o *slot* onde o componente tipo i esta alocado
S: conjunto de *slots*
m: numero de *slots*

O link B representa a seqüência de componentes a ser posicionada e é definido da seguinte forma:

$l_B = [l_B(1), l_B(2),..., l_B(n)]$ onde $l_B(i) \in C$, $l_B(i) \neq l_B(j)$ e $i \neq j$, $A_{i,j} \in \{1, 2, .., n\}$, onde:

gene $l_A(i)$: representa o componente que será montado na i-ésima posição
C: conjunto de componentes
n: numero de componentes

No caso do link A pode acontecer de nem todos os *slots* estarem sendo utilizados, o que depende dos tipos de componentes diferentes (p) utilizados. Portanto caso $p < m$ (número de *slots*), apenas os p primeiros genes serão considerados e teremos m-p genes restantes que neste caso serão redundantes.

O AG ou AM determinará, então, a seqüência ótima, ou próxima da ótima para o arranjo dos carretéis no *slots* e a seqüência de posicionamento dos componentes.

5.6 Aplicação para máquinas Multi Head

Para aplicação do método em máquinas Multi Head, alguns procedimentos adicionais precisarão ser feitos, os quais são descritos a seguir:

5.6.1 Associação de componentes para as cabeças

No caso onde existem mais de uma cabeça é preciso levar em conta duas condições:

- O número de troca de *nozzles* deve ser minimizado, visto que, esta é uma operação extremamente lenta em relação ao tempo de pick-up.
- A carga de trabalho, ou seja, o número de componentes, deve ser dividido o mais igualmente possível entre as cabeças.

Para atender a estas condições os seguintes procedimentos devem ser seguidos:

- Criar conjuntos de componentes que pertencem a um mesmo *nozzle*;
- Usar um AG (ver Lee at al [2]) para distribuir esses conjuntos entre as cabeças de forma que a carga de trabalho seja distribuída o mais uniformemente entre as cabeças.

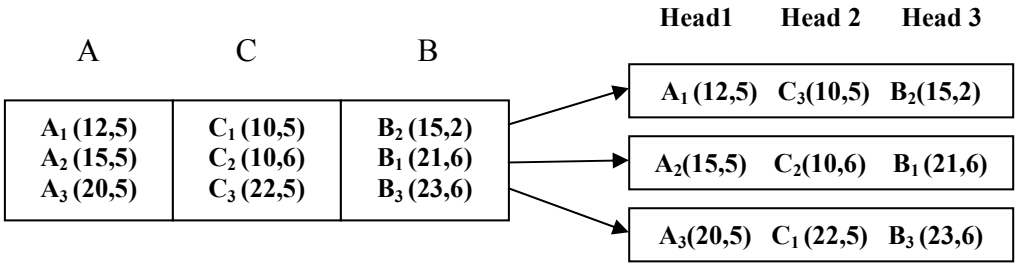
5.6.2 Construção de grupos de componentes

Após a organização dos componentes por *nozzles* e a otimização da carga de trabalho entre as cabeças, deve-se otimizar o número de *pick-ups* simultâneos da máquina. O objetivo é que toda vez que uma SMM de múltiplas cabeças for fazer um pick-up de componentes todas as cabeças capturem um componente. Isto agiliza o tempo de montagem da placa. No trabalho de Lee et al [2] um AG é usado de forma a otimizar o pick-up simultâneo. No final do processamento do AG ele terá criado grupos de componentes de tamanho igual ao número de cabeças da máquina, sendo que cada elemento de um grupo está associado a uma cabeça.

5.7 Aplicação do Método Single Head em máquinas Multi Head

Após a criação dos grupos de componentes, são criados, a partir destes, grupos individuais chamados de *component-cluster*. Considerando um grupo de componentes como um único carretel de componentes e um *component-cluster* como um único componente aplica-se então o mesmo método para máquinas Single Head descrito no item 5.5.

A figura 5.4 abaixo mostra um exemplo de grupos de componentes e *component-clusters*.



Grupo de componentes = carretel *Component-cluster* = componente

Figura 5.4 – Criação de grupos e *Component-Clusters*

São criados então Grupos de componentes como um carretel e *Component-cluster* como um componente para possibilitar a aplicação do método como se a máquina fosse do tipo SIngle Head.

5.8 Aplicação do Algoritmo Memético

A figura 5.5 apresenta o fluxograma do Algoritmo Memético usado neste trabalho apesar de outras implementações poderem ser usadas. A principal característica do AM é o uso de métodos de busca local em adição aos operadores genéticos. Os pontos onde a busca local é inserida podem variar de acordo com o algoritmo proposto. Geralmente a busca local é aplicada

após a formação da população inicial, a aplicação dos operadores genéticos e o término do ciclo de gerações.

A busca local funciona como um refinador das soluções encontradas pelos operadores genéticos, fazendo uma varredura com grau variável de profundidade, na vizinhança das soluções.

A criação da população inicial pode ser feita de diversas formas. Alguns autores criam populações inicialmente com algum grau de otimização, outros, como é o caso deste trabalho, criam populações de forma totalmente randômica usando funções da linguagem de programação usada na implementação do algoritmo.

Nota-se no fluxograma o termo Elite. A Elite são indivíduos, normalmente dois ou três, que são selecionados apenas pelo valor de sua função de aptidão, ou seja, não sofrem a ação dos operadores genéticos.

Os operadores genéticos de cruzamento e mutação são os responsáveis pela criação da próxima geração. Estes métodos possuem diversas propostas de operação. O método de mutação possui a importante função de evitar que o algoritmo caia em mínimos locais enquanto o cruzamento busca a geração de indivíduos mais aptos.

Os critérios de parada mais comumente usados são o número limites de gerações e o tempo limite de processamento. O ponto principal na atuação destes critérios é observar a convergência do algoritmo em torno de uma solução sem que haja melhorias significativas do grau de aptidão.

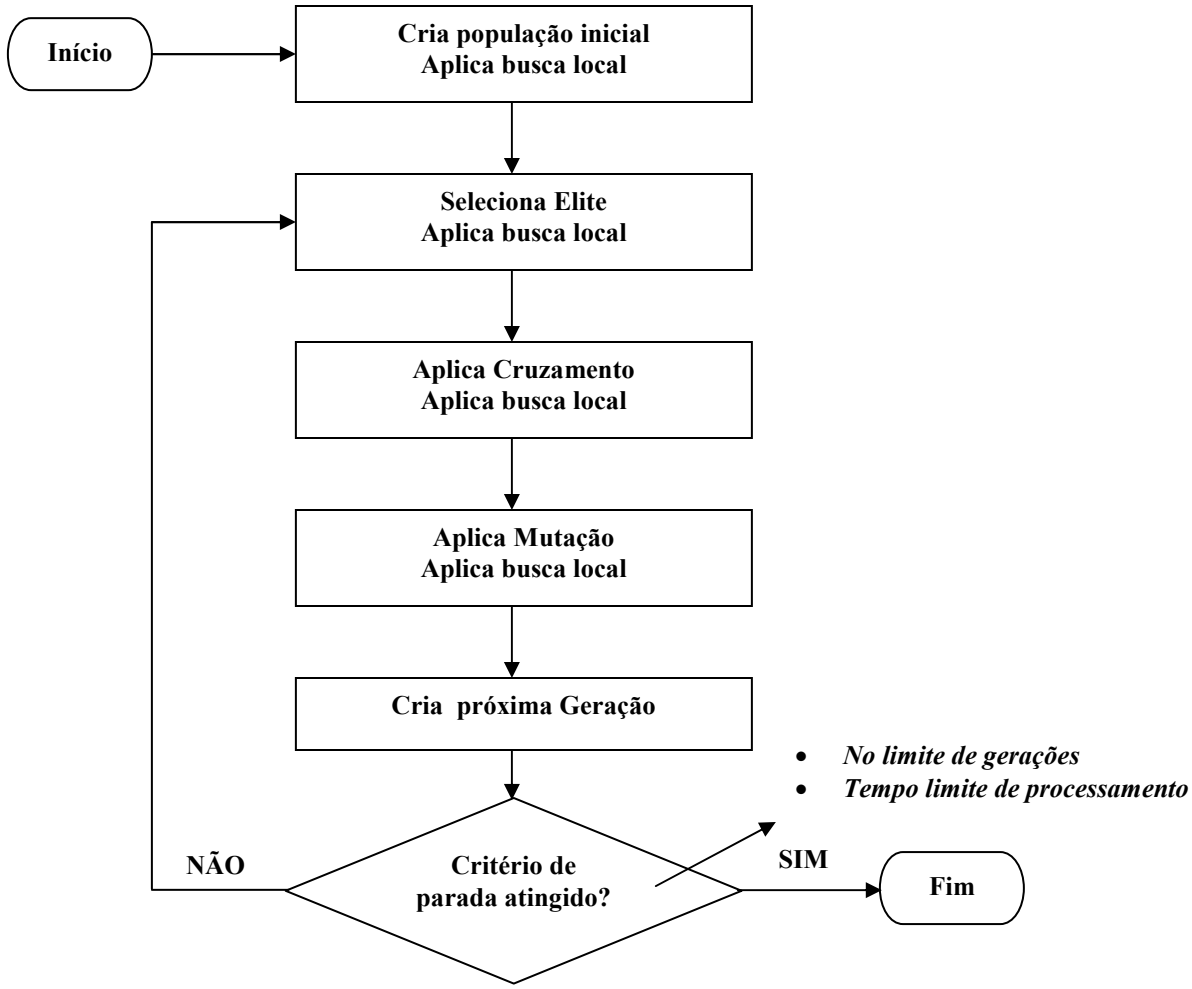


Figura 5.5 – Fluxograma do Algoritmo Memético usado neste trabalho

5.8.1 Algoritmos de busca local utilizados

Existem diversos métodos de busca local que podem ser utilizados em AM. A seguir está um resumo dos métodos utilizados neste trabalho:

5.8.1.1 Busca Tabu

A Busca Tabu é um método adaptativo que usa uma estrutura de memória, e aceita movimentos de piora do valor de Adaptação para escapar de ótimos locais.

Começando com uma solução inicial ***s*** a BT explora, a cada iteração, um subconjunto V da vizinhança $N(s)$ da solução corrente ***s***. O elemento $\boldsymbol{s_0}$ de V com melhor Adaptação nessa região torna-se a nova solução corrente mesmo que $\boldsymbol{s_0}$ seja pior que ***s***, isto é, que $adaptação(s_0) > adaptação(s)$.

Este critério de escolha do melhor vizinho é utilizado para escapar de ótimos locais.

Entretanto, este arranjo pode levar o algoritmo a entrar em *looping*, ou seja, voltar a uma solução que já havia sido gerada.

Para evitar o *looping*, é utilizada uma lista tabu T, a qual é uma memória de soluções já encontradas e que não podem ser consideradas por certo número de gerações.

Durante as iterações cada valor de $\boldsymbol{s_0}$ é comparado com um valor global inicialmente obtido na entrada do método, sendo que o melhor valor é sempre mantido como a solução final a ser devolvida.

5.8.1.2 *Simulated Annealing (SA)*

É uma técnica de busca local probabilística que se baseia em uma analogia com a termodinâmica, simulando o resfriamento de um grupo de átomos aquecidos.

Esta técnica começa sua busca a partir de uma solução inicial qualquer **s**. Em cada iteração é gerado aleatoriamente um único vizinho $\boldsymbol{s_0}$ da solução corrente ***s***.

Considerando $\Delta = Adaptação(s_0) - Fiteness(s)$ as seguintes situações poderão acontecer:

- Se $\Delta < 0$ então $\boldsymbol{s_0}$ passar a ser a nova solução corrente
- Se $\Delta \geq$ então $\boldsymbol{s_0}$ poderá ser aceito mas com uma probabilidade $e^{-\Delta/T.}$

T é um parâmetro chamado de temperatura e que regula a probabilidade de se aceitar soluções de pior custo. Como é uma simulação de resfriamento, a temperatura T assume inicialmente altos valores e vai diminuído a cada iteração. Portanto no início das iterações a probabilidade de aceitar valores piores é maior e, portanto aumenta-se a chance de escapar de ótimos locais. Com o passar do tempo a probabilidade diminui e o algoritmo se comporta como método de descida convergindo para uma solução próxima da ótima.

De forma similar à busca tabu um valor ótimo global é sempre atualizado a cada iteração, o qual será devolvido pelo método como o melhor valor, ou seja, a melhor solução.

Capítulo 06

Resultados Obtidos e Análise

6.1 Parâmetros dos Testes

Para a efetivação dos testes foram gerados programas de máquina de forma randômica, onde o número de tipos de componentes e a quantidade de componentes foram incrementados a cada programa gerado. Os testes foram feitos considerando primeiramente uma máquina do tipo Single Head e posteriormente do tipo Multi Head (com três (3) cabeças).

As simulações foram executadas em um Notebook de 1.66 GHz com WinXP e utilizando o Matlab 7.0.

A máquina virtual usada na simulação possui as seguintes características:

- No de cabeças: 01 ou 03;
- No de tipos de Nozzles disponíveis: 5;
- No de slots: 30;
- Velocidade do braço: 1 unidade de espaço/segundo.

Para a realização dos testes foi considerada uma população inicial de 50 indivíduos e em seguida de 100 indivíduos para avaliar o impacto do tamanho da população no tempo de teste.

O objetivo dos testes foi de comparar o desempenho dos AG e AM sob condições iguais. Portanto os parâmetros genéticos adotados foram fixados nos valores relacionados na tabela 6.1:

Escala de adaptação	Rank	
Função de seleção	Roullete Rheel	
Reprodução	Taxa de crossover 0.8	Taxa de mutação 0.2
Contagem de elites	2	

Tabela. 6.1. Parâmetros Genéticos.

Leia o apêndice para maiores detalhes sobre estes parâmetros.

Os critérios de parada adotados foram os seguintes:

- Numero máximo de gerações: 800;
- Número máximo de gerações sem mudanças: 100;
- Tempo máximo sem mudanças: 2 minutos p/ arranjo de nozzles e cabeças e 5 minutos para seqüência de montagem.

Para os algoritmos de Busca Tabu e Simulated Annealing os parâmetros adotados estão relacionados na tabela 6.2:

Busca Tabu	**Simulated Annealing**
• No de iterações: > 30 • Método de troca de pares	• Temperatura inicial: 0.8 • Passo de decremento: 0.95 • No. De iterações: 20

Tabela. 6.2. Parâmetros dos algoritmos de busca local

6.2 Estrutura dos testes

Os testes foram divididos em duas fases: Simulação e Teste real ou prático.

Em cada um dos testes realizados foi usada a seguinte seqüência:

- Método base usando Algoritmo Genético;
- Método base usando Algoritmo Memético e Tabu Search como busca local;
- Método base usando Algoritmo Memético e Simulated Annealing como busca local.

6.3 Testes Simulados

Os testes simulados foram executados sobre um total de 11 programas de máquina gerados de forma randômica visando testar o poder dos algoritmos envolvidos. Os programas foram gerados a partir de 12 componentes básicos. Para cada programa gerado a quantidade total de componentes da placa foi incrementada, mas sempre a partir dos 12 componentes básicos. Um mesmo tipo de componente poderia, portanto aparecer em diversos pontos da placa, como é no caso real. Para criar o algoritmo AM, em primeiro lugar programei todas as fases do método base de Lee et al [2] no MatLab baseado nas explicações de seu trabalho. Em seguida analisei o *ToolBox* de AG do MatLab e identifiquei os pontos onde inserir as chamadas aos procedimentos de busca local. Com isto consegui programar o AM. O arquivo Matlab modificado foi **c:\matlab7\toolbox\gads\gads\private\stepGA.m** onde o ponto de inserção é destacado na listagem abaixo .

```
function [nextScore,nextPopulation,state] =
stepGA(thisScore,thisPopulation,options,state,GenomeLength,FitnessFcn)
%STEPGA Moves the genetic algorithm forward by one generation
%   This function is private to GA.

%   Copyright 2004 The MathWorks, Inc.
%   $Revision: 1.8 $  $Date: 2004/01/16 16:50:05 $

% how many crossover offspring will there be from each source?
nEliteKids = options.EliteCount;
nXoverKids = round(options.CrossoverFraction * (size(thisPopulation,1) - nEliteKids));
nMutateKids = size(thisPopulation,1) - nEliteKids - nXoverKids;
% how many parents will we need to complete the population?
nParents = 2 * nXoverKids + nMutateKids;

% decide who will contribute to the next generation
% fitness scaling
state.Expectation =
feval(options.FitnessScalingFcn,thisScore,nParents,options.FitnessScalingFcnArgs{:});
% selection. parents are indicies into thispopulation
parents =
feval(options.SelectionFcn,state.Expectation,nParents,options,options.SelectionFcnArgs{:});
```

```
% shuffle to prevent locality effects. It is not the responsibility
% if the selection function to return parents in a "good" order so
% we make sure there is a random order here.
parents = parents(randperm(length(parents)));

[unused,k] = sort(thisScore);
% Everyones parents are stored here for genealogy display
state.Selection = [k(1:options.EliteCount);parents'];

% here we make all of the members of the next generation
eliteKids  = thisPopulation(k(1:options.EliteCount),:);
xoverKids  = feval(options.CrossoverFcn, parents(1:(2 *
nXoverKids)),options,GenomeLength,FitnessFcn,thisScore,thisPopulation,options.CrossoverFcn
Args{:});
mutateKids = feval(options.MutationFcn,  parents((1 + 2 * nXoverKids):end),
options,GenomeLength,FitnessFcn,state,thisScore,thisPopulation,options.MutationFcnArgs{:});

%***********************************
% ALGORITMO MEMÉTICO
% IMPLEMENTANÇÃO DE BUSCA LOCAL (TABU SEARCH)
% 11/04/2007 - MESTRADO EM ENGENHARIA ELÉTRICA
% ELIDELSON CARVALHO

global fAm;
if fAm==1
  eliteKids=tabuts(eliteKids,1,1,FitnessFcn,options);
  xoverKids=tabuts(xoverKids,1,1,FitnessFcn,options);
  mutateKids=tabuts(mutateKids,1,1,FitnessFcn,options);
end
if fAm==2
  eliteKids=sats(eliteKids,20,0.95,0.8);
  xoverKids=sats(xoverKids,20,0.95,0.8);
  mutateKids=sats(mutateKids,20,0.95,0.8);
end

% group them into the next generation
nextPopulation = [ eliteKids ; xoverKids ; mutateKids ];

% score the population
%We want to add the vectorizer if fitness function is NOT vectorized
if strcmpi(options.Vectorized, 'off')
   nextScore = feval(@fcnvectorizer,nextPopulation,FitnessFcn,options.FitnessFcnArgs{:});
else
   nextScore = feval(FitnessFcn,nextPopulation,options.FitnessFcnArgs{:});
end
```

6.4 Teste Real

O teste real foi realizado no setor de Inserção Automática/SMD da SONY Brasil em Manaus/AM em uma máquina Multi Head com 4 cabeças modelo JUKI-2030. Para executar o teste adaptei o sistema de coordenadas usado nos testes simulados para ficarem o mais próximo possível do sistema usado pela máquina. A figura 6.1 mostra a estrutura aproximada usada para otimizar o programa no teste real:

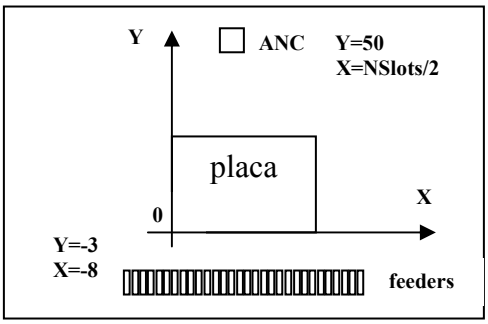


Fig. 6.1. Aproximação do sistema de coordenadas da máquina para o teste real

A modelagem da figura 6.1 foi obtida através de medições e observações dos movimentos da máquina. Os valores negativos Y=-3 e X=-8 significam a posição do primeiro feeder, visto que, de acordo com o sistema de coordenadas criado por mim o ponto (0,0) corresponde ao lado inferior esquerdo da placa. Os pontos Y=50 e X=Nslots/2 correspondem a posição do ANC (automatic nozzle changer) estimada pela minha modelagem. Na realidade a máquina possui 2 (dois) conjuntos de feeders, entretanto, o programa usado no teste usa apenas um conjunto.

Os programas de máquina são normalmente arquivos tipo texto, podendo haver um ou mais arquivos para um modelo. Na figura 6.2 está o programa de máquina usado no teste real. Usando os mesmos dados, criei a minha própria estrutura de programa visando facilitar a programação no Matlab. A criação destas estruturas e a melhor forma de trabalhar os dados do programa é uma decisão do pesquisador ou programador não havendo uma regra específica.

```
--- Production program ---                      18/11/2007 11:45   Page    1

< Placement data/opt. seq. >                    Line name  LINHAS

Program name = teste.H51            18/11/2007 11:45
PWB ID       = KV-29FA210 A
User ID      =
Station      = #1 JUKI1          (2030)
Mode Specification = Parallel Mode

No.(input) CompID      X       Y    Angle  Compo name          Ctrg.   Head Ckt. p  Try  L
01    16   F2        11.00    6.00   0.00 1-216-067-91        L(501)  L1R1    1    No   4
02    14   D1         8.00    5.00   0.00 1-216-809-91        L(503)  L2R2    1 /  No   4
03    22   N2        22.00    3.00   0.00 8-729-920-86        L(504)  L3R3    1    No   4
04    11   B7         9.00    5.00   0.00 1-218-289-91        L(502)  L4R4    1 /  No   4
05    25   F1        18.00    3.00   0.00 1-216-067-91        L(501)  L1R1    1    No   4
06    28   D2        12.00    5.00   0.00 1-216-809-91        L(503)  L2R2    1    No   4
07    27   N1        22.00    4.00   0.00 8-729-920-86        L(504)  L3R3    1 /  No   4
08    33   B5         9.00    6.00   0.00 1-218-289-91        L(502)  L4R4    1    No   4
09    20   F4        19.00    2.00   0.00 1-216-067-91        L(501)  L1R1    1    No   4
10    18   D3        13.00    2.00   0.00 1-216-809-91        L(503)  L2R2    1    No   4
11    15   E4         8.00    3.00   0.00 8-729-920-75        L(504)  L3R3    1    No   4
12    35   B4        13.00    6.00   0.00 1-218-289-91        L(502)  L4R4    1 /  No   4
13    30   F3        24.00    2.00   0.00 1-216-067-91        L(501)  L1R1    1    No   4
14    23   D4        18.00    2.00   0.00 1-216-809-91        L(503)  L2R2    1    No   4
15    19   E2        18.00    5.00   0.00 8-729-920-75        L(504)  L3R3    1    No   4
16     5   B2        16.00    2.00   0.00 1-218-289-91        L(502)  L4R4    1    No   4
17     9   A1        10.00    4.00   0.00 1-162-923-91        L(501)  L1R1    1    No   4
18    32   C2         8.00    4.00   0.00 1-216-186-91        L(503)  L2R2    1    No   4
19    24   E1        18.00    6.00   0.00 8-729-920-75        L(504)  L3R3    1    No   4
20     6   B3        20.00    6.00   0.00 1-218-289-91        L(502)  L4R4    1 /  No   4
21    10   A2        10.00    5.00   0.00 1-162-923-91        L(501)  L1R1    1    No   4
22    36   C4        10.00    4.00   0.00 1-216-186-91        L(503)  L2R2    1    No   4
23    29   E3        24.00    4.00   0.00 8-729-920-75        L(504)  L3R3    1    No   4
24    37   B6        22.00    6.00   0.00 1-218-289-91        L(502)  L4R4    1    No   4
25     2   A4        15.00    2.00   0.00 1-162-923-91        L(501)  L1R1    1    No   4
26    34   C1        12.00    3.00   0.00 1-216-186-91        L(503)  L2R2    1    No   4
27    31   L3         8.00    6.00   0.00 1-125-891-91        L(504)  L3R3    1    No   4
28    39   B1        24.00    6.00   0.00 1-218-289-91        L(502)  L4R4    1 /  No   4
29     1   A3        17.00    6.00   0.00 1-162-923-91        L(501)  L1R1    1    No   4
30    38   C3        13.00    6.00   0.00 1-216-186-91        L(503)  L2R2    1 /  No   4
```

Fig. 6.2. Programa de máquina usado no teste real

6.5 Resultados da simulação para máquina Single Head

6.5.1 Single Head - População de tamanho 50

Programa	Componentes		Genético		Memético+Busca Tabu			Memético+Sim. Annea.		
	Tipos	Total	TEA	TMP	TEA	TMP	Ganho %	TEA	TMP	Ganho %
01	**6**	**12**	**35.37**	**232.14**	**141.45**	**222.83**	**4.17**	**98.64**	**219.71**	**5.66**
02	**8**	**18**	**35.86**	**272.03**	**168.94**	**264.38**	**2.89**	**275.00**	**265.52**	**2.45**
03	**10**	**24**	**89.37**	**397.96**	**302.89**	**396.70**	**0.38**	**358.53**	**394.22**	**0.95**
04	**12**	**30**	**57.70**	**553.29**	**837.98**	**511.91**	**8.08**	**279.98**	**522.08**	**5.98**
05	**14**	**36**	**48.67**	**622.00**	**724.20**	**572.19**	**8.70**	**398.69**	**592.55**	**4.97**
06	**16**	**42**	**152.50**	**723.05**	**1078.31**	**700.58**	**3.20**	**640.90**	**705.87**	**2.43**
07	**18**	**48**	**84.90**	**910.22**	**1722.62**	**875.07**	**4.02**	**410.44**	**918.90**	**0.00**
08	**20**	**54**	**148.39**	**931.29**	**1562.70**	**929.74**	**0.17**	**1044.53**	**920.09**	**1.22**
09	**22**	**60**	**108.48**	**1055.98**	**1763.47**	**1019.22**	**3.60**	**883.98**	**1070.02**	**0.01**
10	**24**	**66**	**135.58**	**1271.38**	**2804.12**	**1213.59**	**4.76**	**1240.57**	**1212.29**	**4.87**
11	**26**	**72**	**184.17**	**1364.52**	**2920.55**	**1310.20**	**4.14**	**1280.67**	**1300.15**	**4.95**

Tab. 6.3 Tempos de execução dos algoritmos, tempos de montagem das placas e melhoria percentual obtida para população de 50 indivíduos em máquina tipo Single Head

Os resultados apresentados na tabela 6.3 são mais bem entendidos através dos gráficos mostrados nas figuras 6.3 e 6.4.

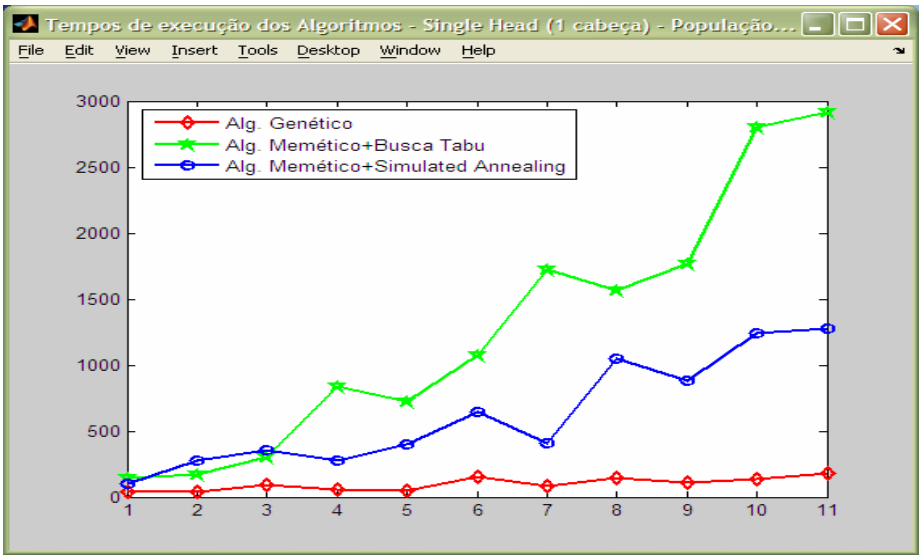


Fig. 6.3. Gráfico dos tempos de execução dos algoritmos para população de 50 indivíduos em máquina tipo Single Head

O gráfico da figura 6.3 mostra claramente que o tempo de execução o AG foi mais rápido do que o tempo do AM, visto que o AG não faz uso da busca local. Também se observa que o método de busca local *Simulated Annealing* foi mais rápido do que a Busca Tabu.

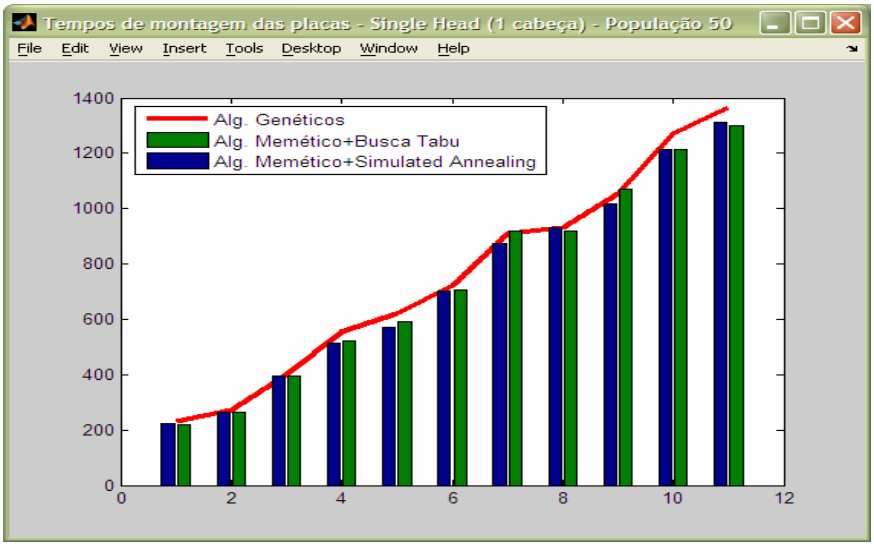


Fig. 6.4. Gráfico dos tempos de montagem das placas para população de 50 indivíduos em máquina tipo Single Head

O gráfico da figura 6.4 mostra que o AM superou o AG no tempo de montagem da placa e também que o desempenho do AM cresce a medida que o numero de componentes também cresce. O leitor pode observar que os métodos de busca local se alternaram na performance, de onde se pode concluir que houve um equilíbrio entre ambos.

6.5.2 Single Head - População de tamanho 100

Programa	Componentes		Genético		Memético+Busca Tabu			Memético+Sim. Annea.		
	Tipos	total	TEA	TMP	TEA	TMP	Ganho %	TEA	TMP	Ganho %
01	6	12	32.51	205.04	144.81	205.11	0.0	292.36	205.18	0.00
02	8	18	41.44	306.50	327.21	297.34	3.08	649.09	298.21	2.78
03	10	24	38.09	432.60	446.06	417.75	3.55	484.87	419.54	3.11
04	12	30	98.28	520.42	437.17	513.33	1.38	338.89	524.44	0.0
05	14	36	107.30	634.96	1707.05	622.72	1.96	702.17	617.13	2.89
06	16	42	139.48	764.16	1517.06	763.90	0.03	670.51	780.49	0.02
07	18	48	160.20	840.41	2942.97	771.87	8.88	4035.80	795.25	5.68
08	20	54	300.62	1004.21	6729.23	957.16	4.91	3419.23	959.52	4.66
09	22	60	238.87	1042.47	6223.59	1004.59	3.77	1845.31	1034.14	0.8
10	24	66	249.23	1239.72	2847.34	1210.76	2.39	4400.86	1234.46	0.43
11	26	72	645.45	1290.21	3337.56	1330.99	-0.03	5883.83	1290.52	0.0

Tab. 6.4. Tempos de execução dos algoritmos, tempos de montagem das placas e melhoria percentual obtida para população de 100 indivíduos em máquina tipo Single Head

A tabela 6.4 apresenta os resultados para máquina Single Head agora para uma população de 100 indivíduos. Os resultados são mais bem entendidos através dos gráficos das figuras 6.5 e 6.6.

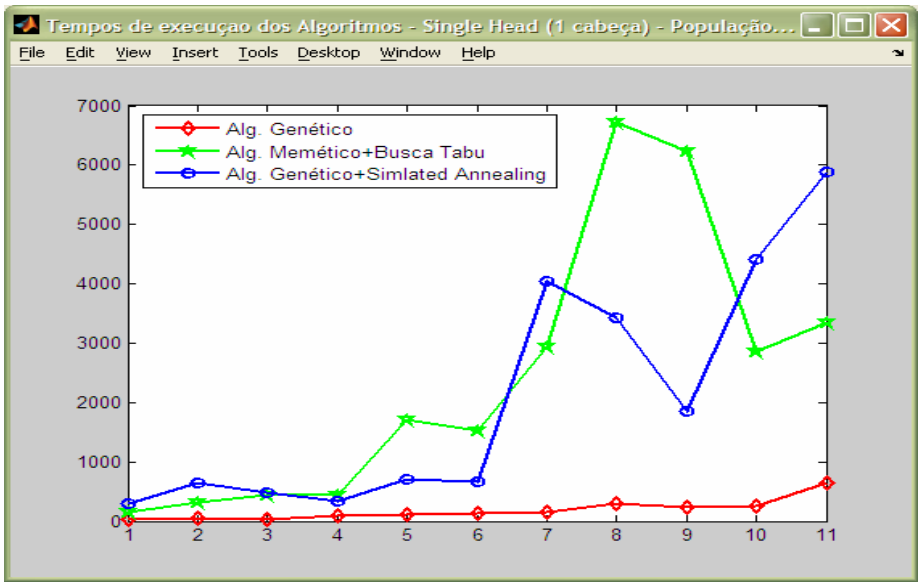


Fig. 6.5. Gráfico dos tempos de execução dos algoritmos para população de 100 indivíduos em máquina tipo Single Head

O gráfico da figura 6.5 repete o melhor desempenho do AG em relação ao AM na questão do tempo de processamento como aconteceu com a população de 50 indivíduos. Entretanto se observou um equlíbrio no desempenho dos métodos de busca local *Simulated Annealing* e Busca Tabu.

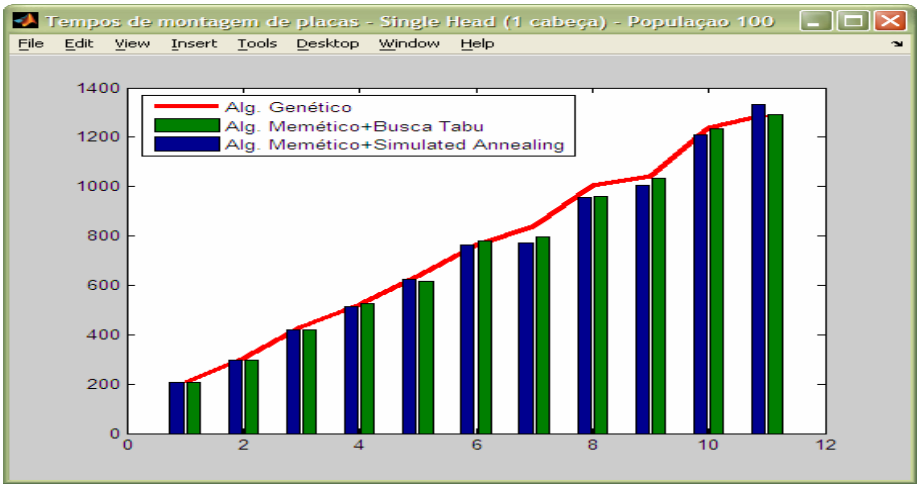


Fig. 6.6. Gráfico dos tempos de montagem das placas para população de 100 indivíduos em máquina tipo Single Head

O gráfico da figura 6.6 mostra novamente o melhor desempenho do AM no tempo de montagem. Entretanto nota-se que os resultados são praticamente equivalentes para poucos componentes. Observa-se também o desempenho um pouco melhor da Busca Tabu em relação à *Simulated Annealing*.

6.6 Resultados da simulação para máquina Multi Head

6.6.1 Multi Head - População de tamanho 50

Programa	Componentes		Genético		Memético+Busca Tabu			Memético+Sim. Annea.		
	Tipos	total	TEA	TMP	TEA	TMP	Ganho %	TEA	TMP	Ganho %
01	6	12	13.20	188.36	106.12	175.03	7.62	130.90	175.94	7.06
02	8	18	21.76	248.06	99.21	241.43	2.75	99.56	233.91	6.05
03	10	24	19.67	299.90	123.56	293.77	2.01	145.68	288.19	4.06
04	12	30	28.20	364.66	196.59	341.81	6.69	148.59	358.02	1.85
05	14	36	50.70	438.11	244.51	422.40	3.72	163.98	434.91	0.74
06	16	42	83.51	596.40	364.53	630.86	-0.05	293.06	609.94	-0.02
07	18	48	66.95	678.94	385.75	608.44	11.60	485.07	590.92	14.90
08	20	54	131.84	938.97	435.03	711.38	32.00	724.60	735.54	27.66
09	22	60	119.07	791.94	494.46	768.97	2.99	396.00	887.10	0.10
10	24	66	104.48	1038.27	706.40	987.41	5.15	655.95	925.78	12.15
11	26	72	77.00	1076.44	588.10	1119.07	-0.03	594.20	958.38	12.31

Tab. 6.5. Tempos de execução dos algoritmos, tempos de montagem das placas e melhoria percentual obtida para população de 50 indivíduos em máquina tipo Multi Head com 03 cabeças

A tabela 6.5 apresenta os dados para uma máquina Multi Head com população de 50 indivíduos. A s informações da tabela são destacadas nos gráficos das figuras 6.7 e 6.8.

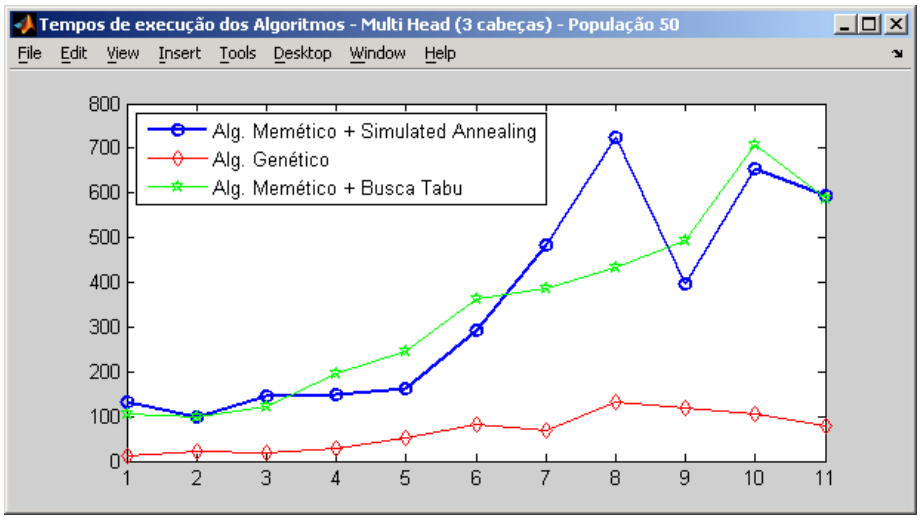


Fig 6.7. Gráfico dos tempos de execução dos algoritmos para população de 50 indivíduos em máquina tipo Multi Head com 3 cabeças

O gráfico da figura 6.7 mostra que o aumento do número de cabeças reduziu bastante o tempo de processamento, enquanto os métodos de busca local se alternaram no desempenho.

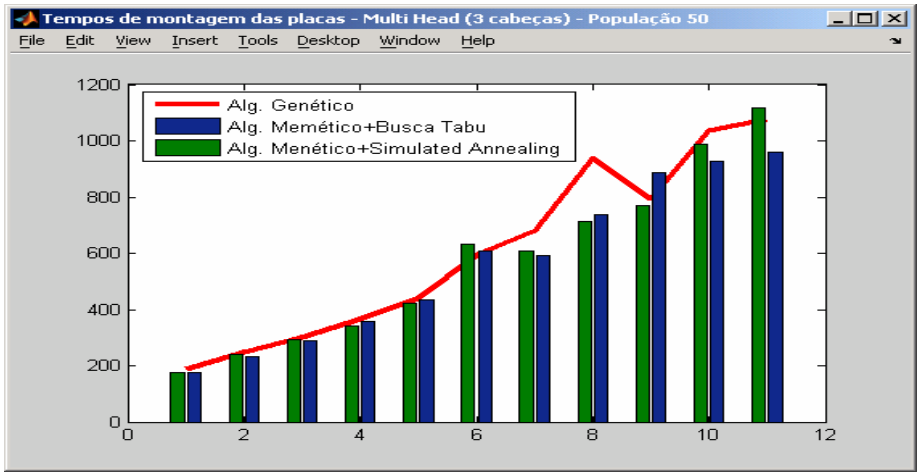


Fig. 6.8 Gráfico dos tempos de montagem das placas para população de 50 indivíduos em máquina tipo Multi Head com 3 cabeças

O gráfico da figura 6.8 mostra que o aumento do número de cabeças introduziu uma melhor do AG em relação ao caso Single Head., entretanto o AM ainda obteve melhor desempenho.

6.6.2 Multi Head - População de tamanho 100

Programa	Componentes		Genético		Memético+Busca Tabu			Memético+Sim. Anneal.		
	Tipos	Total	TEA	TMP	TEA	TMP	Ganho %	TEA	TMP	Ganho %
01	6	12	28.40	137.21	110.28	126.15	8.77	188.48	142.69	-0.03
02	8	18	26.95	262.05	380.57	258.02	1.56	247.42	245.92	6.56
03	10	24	38.15	313.65	483.75	359.98	-0.14	336.31	344.14	-0.09
04	12	30	58.46	391.55	479.45	393.17	0.0	279.42	380.83	2.81
05	14	36	49.39	512.39	721.85	475.85	7.68	460.59	488.45	4.90
06	16	42	63.59	501.74	485.10	529.81	-0.05	429.00	425.50	17.92
07	18	48	102.39	760.27	790.28	698.79	8.80	897.84	710.18	7.05
08	20	54	123.06	725.75	664.25	678.84	6.91	538.75	707.73	2.55
09	22	60	83.84	752.24	650.95	772.51	-0.02	974.51	743.39	1.19
10	24	66	195.56	876.80	1137.21	858.91	2.08	762.89	876.09	0.00
11	26	72	119.23	1060.50	1377.21	1013.08	4.68	1551.56	987.39	7.40

Tab. 6.6. Tempos de execução dos algoritmos, tempos de montagem das placas e melhoria percentual obtida para população de 100 indivíduos em máquina tipo Multi Head com 03 cabeças

Os gráficos da tabela 6.6 são mostrados nas figuras 6.9 e 6.10.

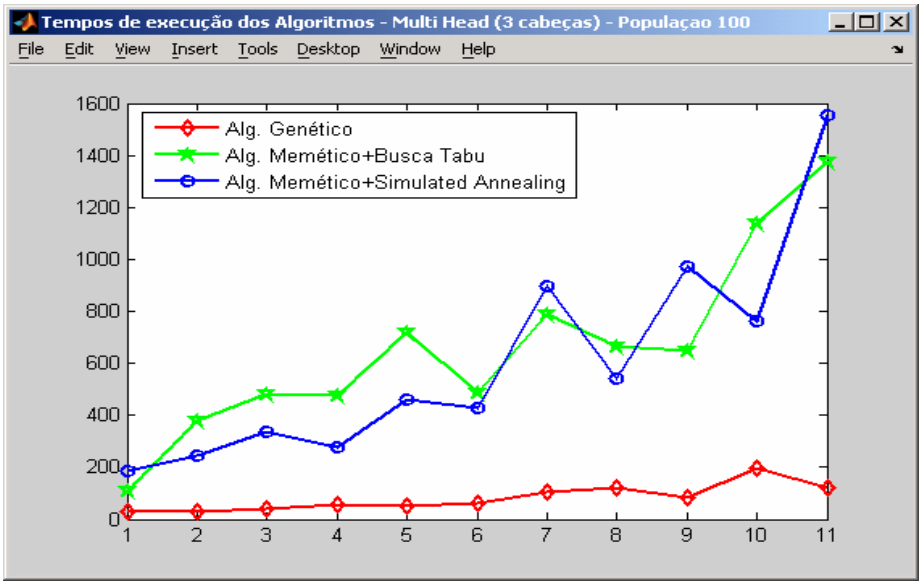


Fig. 6.9. Gráfico dos tempos de execução dos algoritmos para população de 100 indivíduos em máquina tipo Multi Head com 3 cabeças

O gráfico da figura 6.9 novamente apresenta o melhor desempenho do AG no tempo de processamento. A equivalência dos métodos de busca local também se mantiveram.

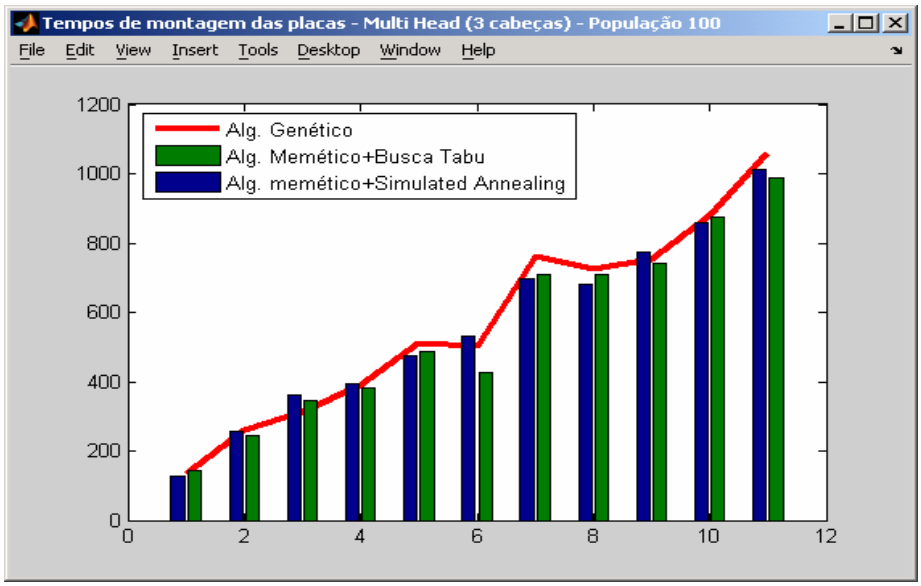


Fig. 6.10. Gráfico dos tempos de montagem das placas para 100 indivíduos em máquina tipo Multi Head com 3 cabeças

O gráfico da figura 6.10 confirma os resultados anteriores. É interessante notar que de forma análoga à máquina de uma cabeça, o aumento da população inicial influi diretamente no melhor desempenho do AM.

6.7 Resultados do Teste Real

O teste real foi feito para comparar o desempenho dos Algoritmos Genéticos e Meméticos em um equipamento real. Além disso, foi interessante comparar o desempenho dos Algoritmos em relação à solução apresentada pelo software HLC fornecido pelo fabricante da máquina.

A seguir estão listadas em forma de tabela as estruturas dos programas utilizados no teste prático.

ciclo	**Componentes**				**Heads**				**Feeders**			
1	b7	F2	n2	d1	2	1	4	3	18	15	13	14
2	b5	F1	n1	d2	2	1	4	3	18	15	13	14

3	b4	F4	j1	d3	2	1	4	3	18	15	22	14
4	b2	F3	j3	d4	2	1	4	3	18	15	22	14
5	b3	A1	j2	c2	2	1	4	3	18	20	22	21
6	b6	A2	l3	c4	2	1	4	3	18	20	12	21
7	b1	A4	l1	c1	2	1	4	3	18	20	12	21
8	g1	A3	l4	c3	2	1	4	3	23	20	12	21
9	e4	E2	l2	c5	4	4	4	3	19	19	12	21
10	e1	E3	m3		4	4	4		19	19	16	
11	m2	M1	I1		4	4	4		16	16	09	
12	h1	K1			4	4			11	27		

Tab. 6.7. Programa gerado pelo Algoritmo Genético

A tabela 6.7 apresenta os componentes, as cabeças e o feeders utilizados em cada ciclo do programa geardo pelo AG.

ciclo	**Componentes**				**Heads**				**Feeders**			
1	f2	D1	N2	b7	1	3	4	2	18	19	21	20
2	f1	D2	N1	b5	1	3	4	2	18	19	21	20
3	f4	D3	E4	b4	1	3	4	2	18	19	10	20
4	f3	D4	E2	b2	1	3	4	2	18	19	10	20
5	a1	C2	E1	b3	1	3	4	2	12	11	10	20
6	a2	C4	E3	b6	1	3	4	2	12	11	10	20
7	a4	C1	l3	b1	1	3	4	2	12	11	13	20
8	a3	C3	l1	g1	1	3	4	2	12	11	13	9
9	l2	C5	l4	m3	4	3	4	4	13	11	13	17
10	m2		M1	j1	4		4	4	17		17	22
11	j3		j2	i1	4		4	4	22		22	14
12	h1		K1		4		4		23		25	

Tab. 6.8. Programa gerado pelo Algoritmo Memético com Busca Tabu

A tabela 6.8 apresenta os componentes, as cabeças e o feeders utilizados em cada ciclo do programa geardo pelo AM+Busca Tabu, onde se observa a diferença nos ciclos em relação ao programa gerado pelo AG.

ciclo	Componentes				Heads				Feeders			
1	B7	N2	D1	F2	2	4	3	1	19	16	15	17
2	B5	N1	D2	F1	2	4	3	1	19	16	15	17
3	B4	L3	D3	F4	2	4	3	1	19	10	15	17
4	B2	L1	D4	F3	2	4	3	1	19	10	15	17
5	B3	L4	C2	A1	2	4	3	1	19	10	9	8
6	B6	L2	C4	A2	2	4	3	1	19	10	9	8
7	B1	E4	C1	A4	2	4	3	1	19	20	9	8
8	G1	E2	C3	A3	2	4	3	1	13	20	9	8
9	E3	E1	C5	M3	4	4	3	4	20	20	9	12
10	M2	M1		J1	4	4		4	12	12		21
11	J3	J2		I1	4	4		4	21	21		14
12	H1	K1			4	4			24	22		

Tab. 6.9. Programa gerado pelo Algoritmo Memético com Simulated Annealig

A tabela 6.9 apresenta os componentes, as cabeças e o feeders utilizados em cada ciclo do programa geardo pelo AM+Simulated Annealing, onde se observa a diferença nos ciclos em relação ao programa gerado pelo AG e pelo AM+Busca Tabu.

ciclo	Componentes				Heads				Feeders			
1	A4	F3	G1	b1	1	1	2	2	36	38	40	42
2	A1	F4	B6	b3	1	1	2	2	36	38	42	42
3	A2	F2	B2	b4	1	1	2	2	36	38	42	42
4	A3	F1	B5	b7	1	1	2	2	36	38	42	42
5	J1	M3	C2	d4	4	4	3	3	20	22	24	26
6	J3	M2	C4	d3	4	4	3	3	20	22	24	26
7	J2	M1	C1	d2	4	4	3	3	20	22	24	26
8	E4	E3	C5	c3	4	4	3	3	16	18	24	24
9	E2	E1		d1	4	4		3	16	18		26
10	L4	L3			4	4			08	10		
11	L2	L1			4	4			08	10		
12	N2	N1			4	4			48	50		
13	K1	I1			4	4			04	06		
14	H1				4				02			

Tab. 6.10. Programa gerado pelo software HLC fornecido pelo fabricante da máquina.

A tabela 6.10 apresenta os componentes, as cabeças e o feeders utilizados em cada ciclo do programa gerado pelo software HLC do fabricante da máquina.

Os resultados dos testes práticos estão tabelados a seguir:

Algoritmos	**Resultados dos Algoritmos para teste real**									
	CPP	**TN**	**VANC**	**NF**	**PS4**	**PS3**	**PS2**	**PNS**	**Tempo(s)**	**Ganho%**
Genético	**12**	**2**	**1**	**14**	**9**	**2**	**1**	**0**	**31,6**	**- 12,05**
Memético(BT)	**12**	**2**	**1**	**14**	**9**	**2**	**1**	**0**	**30,8**	**- 9,21**
Memético(SA)	**12**	**2**	**1**	**14**	**9**	**2**	**1**	**0**	**29,5**	**- 4,60**
HLC	**14**	**4**	**1**	**17**	**8**	**1**	**4**	**1**	**28,2**	**0.0**

Tab. 6.11. Tabela de resultados do teste prático

Na tabela 6.11 vemos que o método AM+SA foi o que obteve desempenho mais próximo do HLC. Nota-se que o AM obteve um desempenho bem melhor do que o AG, mostrando o poder refinador do uso de métodos de busca local.

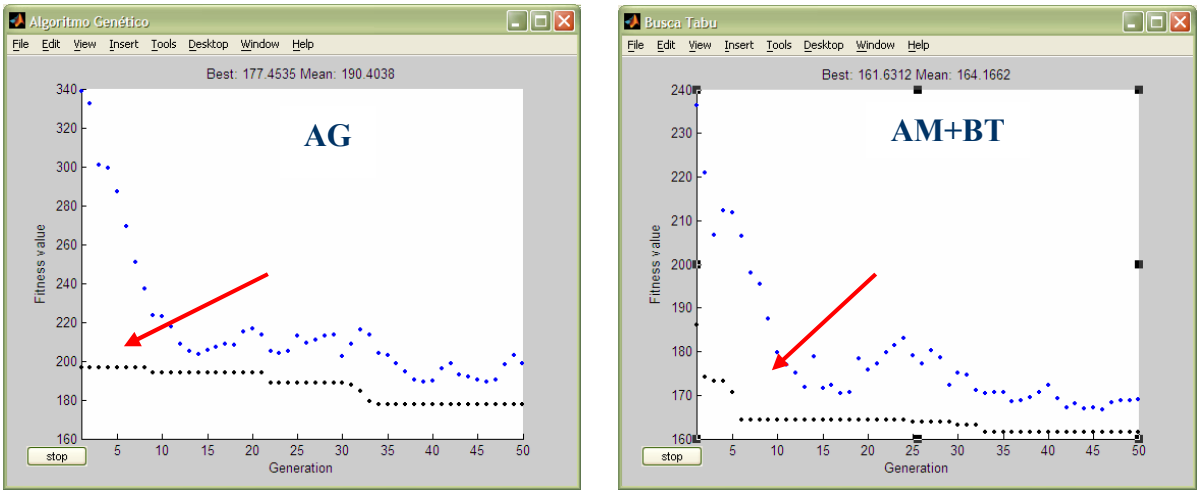
É interessante notar que o HLC apresentou mais ciclos e troca de nozzles e mesmo assim foi mais rápido. Este desempenho melhor do HLC justifica-se principalmente pela permissão do uso de um mesmo componente em feeders diferentes, o que não é permitido pelo algoritmo usado neste trabalho.

Na tabela acima temos:

CPP - número total de ciclos de *pick & place*
TN : - total de trocas de *nozzle*
VANC - total de visitas ao ANC
NF: : - quantidade de alimentadores utilizados
PS4: : - número de ciclos com *pickup* simultâneo de 4 cabeçotes
PS3: : - número de ciclos com *pickup* simultâneo de 3 cabeçotes
PS2: : - número de ciclos com *pickup* simultâneo de 2 cabeçotes
PNS: : - número de ciclos sem nenhum *pickup* simultâneo
Tempo - tempo cronometrado para a montagem dos componentes na placa.
Ganho - percentual de melhoria em relação à solução do HLC.

6.8 Exemplo de convergência dos algoritmos

Os gráficos das figuras 6.11.a e 6.11.b mostram como o Algoritmo Memético com Busca Tabu consegue convergir mais rápido para a região de valores próximos do ótimo em comparação ao Algoritmo Genético.



(a) (b)

Fig. 6.11. Exemplo de convergência para Algoritmo Genético e Algoritmo Memético com Busca Tabu para um mesmo conjunto de dados

6.9 Exemplos de seqüências de montagem obtidas

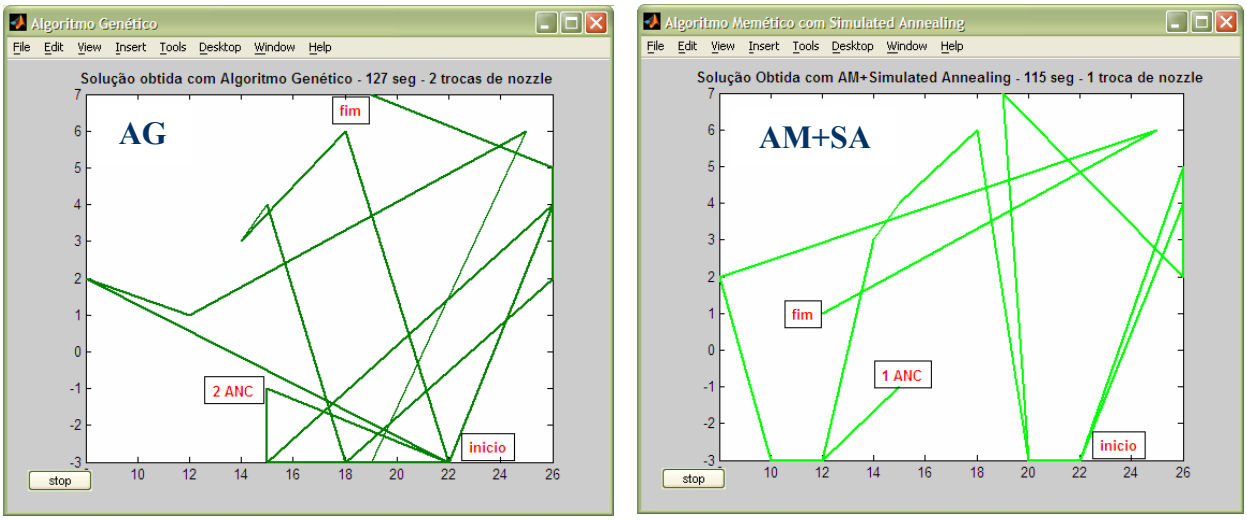


Fig 6.12. Exemplo de seqüência de montagem obtidas com Algoritmo Genético e Algoritmo Memético com Simulated Annealing

6.10 Conclusões da análise

Na análise dos resultados procuramos comparar o desempenho dos algoritmos em relação aos seguintes pontos:

- O tempo de execução ou de processamento dos algoritmos;
- O tempo de montagem das placas;
- O tempo de montagem das placas obtido com os Algoritmos Memético e Genético em relação ao software HLC fornecido pelo fabricante da máquina.

Os resultados obtidos na simulação mostram que os tempos de execução dos algoritmos dependem do número de componentes, da disposição desses componentes na placa, do tamanho da população e dos critérios de parada. Os resultados apontam uma grande diferença entre os tempos de execução do AG e do AM à medida que o número de componentes cresce. Isto é decorrência, sem dúvida, da presença dos algoritmos de busca local. No caso de máquinas Single Head estes tempos crescem ainda mais e podem chegar a algumas horas de processamento a partir de 50 componentes para populações acima de 100 indivíduos. Os resultados também mostram um melhor desempenho no tempo de execução dos algoritmos da composição AM+SA em relação a AM+BT. Foi observado que a maior demora da BT a cada ciclo deve-se ao fato que ela faz uma busca mais exaustiva na vizinhança de uma solução do que a SA, o que justifica um maior tempo de processamento.

Os resultados mostram que os AM conseguem efetivamente melhorar os tempos de montagem das placas com uma ligeira vantagem do método AM+BT em relação ao método AM+SA. Estas melhorias chegaram próximas aos 30% no caso do uso de máquinas Multi Head, como visto nos gráficos 6.7 e 6.8, sendo isto um ganho significativo neste tipo de problema, onde

qualquer diminuição significativa de tempo irá ser evidenciada com o aumento de produtividade da linha de produção ao final de turnos de trabalho.

Em todos os gráficos também se observa que nos piores casos pelo menos um dos métodos AM, ou seja, AM+BT ou AM+SA, consegue ser mais eficiente que o AG. Quando o AG supera um dos métodos AM, isto acontece devido à natureza estocástica do método de seleção onde pode acontecer que os operadores genéticos não sintam a influência da busca local.

Em relação ao teste prático o mesmo foi importante para validar os resultados dos testes simulados, pois os algoritmos puderam ser testados em uma máquina real. Novamente os resultados obtidos, vistos na tabela 6.11, mostraram a melhor eficiência do Algoritmo Memético em relação ao Genético para o problema em questão. Outro resultado importante é que o Algoritmo Memético conseguiu um tempo de execução bem próximo ao do software HLC fornecido pelo fabricante da máquina levando-se em consideração que o HLC possui a vantagem de admitir feeders com componentes repetidos, o que não é aceito pelo método no qual este trabalho está baseado.

Os bons resultados obtidos pelo Algoritmo Memético, entretanto, possuem um custo, ou seja, o aumento do tempo de processamento imposto pelos métodos de busca local, principalmente para máquinas Single Head. Este fato foi observado tanto nos testes simulados quanto nos práticos. Mas, Como a tendência da indústria é optar por máquinas Multi Head, é possível que este tipo de custo não venha a comprometer o uso do método.

Capítulo 07

Conclusões e Pesquisas Futuras

O objetivo deste trabalho era investigar a utilização de Algoritmos Meméticos no importante problema da otimização do tempo de montagem de componentes eletrônicos SMD em placas de circuito impresso.

A motivação para este estudo foi o fato de não ter sido encontrada na literatura estudada referências para uso deste algoritmo na resolução deste problema. Além disso, os diversos trabalhos que utilizam com sucesso Algoritmos Genéticos neste problema de otimização, conduziram a uma interessante comparação do desempenho destes dois algoritmos, visto que, a literatura apresenta diversos casos de resultados significativamente melhores dos Algoritmos Meméticos em relação aos Genéticos em diversas áreas de aplicação.

O trabalho de Lee et al [2] que faz uso intenso de Algoritmos Genéticos foi importante neste estudo, pois apresenta resultados superiores aos obtidos pelos algoritmos comumente usados na indústria, o que serve como validação dos resultados obtidos com o método dos Algoritmos Meméticos. Além disso, os detalhes apresentados no trabalho permitiram sua implementação no Matlab.

Os algoritmos de busca local utilizados tiveram um desempenho aceitável e mostraram-se adequados para este tipo de problema, alternando-se na obtenção de melhores resultados.

Os resultados obtidos mostraram que o método baseado no AM consegue efetivamente reduzir o tempo de montagem das PCB's chegando a reduções próximas de 30% em relação ao método baseado no AG.

Apesar do aumento no tempo computacional com a utilização dos algoritmos de busca local, o qual foi sentido principalmente nas máquinas tipo Single Head, é possível minimizar este problema usando métodos de busca local mais eficientes e por fim processadores mais velozes, programas escritos em linguagens apropriadas como o C++ e máquinas tipo Multi Head, as quais são as mais comumente usadas na industria. Além disso, é perfeitamente aceitável se gastar algumas horas de processamento, mas ao final obter-se excelentes soluções de seqüências de montagem de PCB's.

Portanto o método proposto baseado em Algoritmo Memético, se apresenta como uma promissora ferramenta alternativa principalmente aos diversos métodos baseados em AG encontrados na literatura.

Pesquisas Futuras

1. **Testar outros métodos de Busca Local**

   Existem diversos outros métodos de busca local que podem ser testados em conjunto com os Algoritmos Meméticos buscando tanto a melhoria dos tempos de montagem quanto dos tempos de processamento do algoritmo.

2. **Usar o método baseado em AM como refinador do AG**

   A literatura cita que os Algoritmos Genéticos são um eficiente método de otimização, entretanto as soluções encontradas são geralmente sub-ótimas podendo ser refinadas aplicando-se outro método de otimização na população final gerada pelo Algoritmo Genético. Fica então um interessante proposta de usar Algoritmo Memético como este método refinador.

3. **Usar o método baseado em AM nos diversos problemas de linhas de produção de PCB's.**

Como foi citado no capítulo 1 em uma linha de produção de PCB's existem os seguintes pontos de otimização agrupados de forma hierárquica:

- Problema de PCP (planejamento e controle de produção);
- Problema de balanceamento de linha;
- Problema de otimização de set-up times (tempo de preparação de máquinas).

Surge então uma interessante linha de pesquisa para estudar o uso de Algoritmos Meméticos na resolução destes problemas de otimização.

4. **Testar o uso de Algoritmos Transgenéticos.**

Os algoritmos transgenéticos são algoritmos evolucionários e se baseiam no processo de simbiogênese (Goldbarg [18]). É uma teoria evolucionária onde indivíduos de diferentes espécies se unem para formar um novo indivíduo. Ela enfatiza mais os efeitos positivos resultantes das inter-relações entre indivíduos do que a sobrevivência e reprodução do mais apto.

Referências Bibliográficas

[1] Carlos Henrique Craveiro. **Otimização do posicionamento e da montagem automática de componentes eletrônicos através de um algoritmo genético**. FT, COPPE, UFRJ, 2005.

[2] Wonsik Lee, Sungham Lee, Beomhee Lee e Youngdae Lee. **A Genetic Optimization Approach to Operation of a Multi Head Surface Machine**. IEICE Trans.Fundamentals, v. E83-A, n. 9, pp. 1748-1756, Sep. 2000.

[3] Zeng You-jiao. **Hierarchical planning for a surface mounting machine placement.** Journal of Zhejiang University;

[4] Edmund K. Burke, Peter I. Cowling, Ralph Keuthen. **New models and heuristics for component placement in printed circuit board assembly.** University of Nottingham;

[5] N.S. Ong, W.C.Tan. **Sequence placement planning for high-speed PCB assembly machine.** School of Mechanical and Production Engineering, Singapore;

[6] L.P.Khoo, K.M.Loh. **A Genetic Algorithms Enhanced Planning System for Surface Mount PCB assembly.** School of Mechanical and Production Engineering, Singapore;

[7] Masri Ayob, Peter Cowling, Graham Kendall. **Optimization for Surface Mount Placement Machines.** University of Nottingham;

[8] Mika Johnsson, Jouni Smed. **Observations on PCB Assembly Optimization.** University of Turku, Finlândia;

[9] Masry Ayob, Graham Kendall. **A triple objective function with a Chebychev dynamic pick-and-place point specification approach to optimize the surface mount placement machine.** University of Nottingham;

[10] Kymberly P. Ellis, Fernando J. Vittes, John E. Kobza. **Optmizing the Performance of a Surface Mount Placement Machine.** IEEE members;

[11] Scott Robert Ladd. **Genetic Algorithms in C++.** M&T Books;

[12] Melanie Mitchell. **An Introduction to Genetic Algorithms.** MIT press;

[13] Alexandre de Sousa Mendes. **Algoritmos meméticos aplicados aos problemas de sequenciamento em máquinas.** Tese de mestrado, Unicamp, 1999;

[14] Shawki Areibi, Medhat Moussa, Hussein Abdullah. **A Comparison of Genetic/Memetic Algorithms and other heuristic search techniques.** University of Guelph, Ontario, CANADA;

[15] Peter Merz, Bernd Freisleben. **Genetic Local Search for the TSP.** University Siegen, Alemanha;

[16] Natalio Krasnogor. **Memetic Algorithms.** Granada, Espanha.

[17] Ender Ozcan, Murat Erenturk. **A Brief Review of Memetic Algorithms for Solving Euclidean 2D Traveling Salesman Problem.** Yeditepe University,Istanbul, Turkey.

[18] Marco César Goldbarg, Elizabeth Ferreira Gouvêa Goldbarg, Francisco Dantas de Medeiros Neto. **Algoritmos evolucionários na determinação da configuração de custo mínimo de sistemas de co-geração de energia com base no gás natural.** Depto. de Informática e Matemática Aplicada (DIMAp)Universidade Federal do Rio Grande do Norte (UFRN); Natal – RN.

[19] Ball, M., magazine. **Sequencing of Insertion in Printed Circuit Board Assembly***, Operations Research,* v. 36, n. 2, pp.192-210, 1989.

[20] Leipala, T., Nevalainen, O. "Optimization of the Movements of a Component Placement Machine"*,* European Journal of Operational Research, v. 38, pp.167-177, 1989.

[21] Kumar, R. and Li, H. **"Integer Programming Approach to Printed Circuit Board Assembly Time Optimization"**, IEEE Trans. Components, Packaging and Manufacturing Technology - Part B, v. 18, n. 4, pp.720-727, Nov. 1995.

[22] M. C. Leu, H. Wong and Z. Ji, **"Planning of component placement/insertion sequence and feeder setup in PCB assembly using genetic algorithm"**, Journal of Electronic Packaging, Transactions ASME, 115, pp. 424–432, 1993.

[23] Wang, C., Ho, L.H., Fu, H.I. and Su, Y.C (1997), **"A magazine assignment heuristic for robotic assembly using the dynamic pick-and-place approach"** International journal of industrial Engineering, Vol 4, No. 1, pp 24-33.

[24] HO, W., JI, P., **"A Hybrid Genetic Algorithm for Component Sequencing and Feeder Arrangement"**, Journal of Intelligent Algorithm, v.15, pp.307-315, 2004.

[25] Marcone Jamilson Freitas Souza., **"Notas de aula da disciplina Inteligência Computacional para Otimização"**, Universidade Federal de Ouro Preto, 2005.

Apêndice

Algoritmos Genéticos e Meméticos

Algoritmos Genéticos e Meméticos são métodos computacionais de otimização, chamados de evolucionários, porque são inspirados na teoria da evolução natural das espécies, onde em uma determinada população, o cruzamento genético gera novos indivíduos com melhor capacidade de adaptação ao meio em que vivem.

As principais vantagens dos algoritmos Genéticos e Meméticos são listadas a seguir:

- São robustos para problemas altamente não lineares;
- Normalmente conseguem evitar a queda em mínimos locais.

No contexto computacional os indivíduos são geralmente representados por arrays ou vetores, cujos elementos podem ser de diversos tipos, como por exemplo, números binários, strings, e são chamados de cromossomos. Por sua vez cada posição de um array (cromossomo) é chamada de gene.

Cada array (cromossomo) representa uma solução para o problema em questão no chamado espaço de soluções. Um método especial (procedimento) chamado de função de Fitness, ou de custo, quando aplicado em um indivíduo (cromossomo), retorna um valor que indica quão bom ele é como solução para o problema em questão.

Métodos especiais (procedimentos) chamados de Operadores Genéticos recebem um ou mais cromossomos (chamados de pais) como parâmetros de entrada e fornecem novos cromossomos (chamados de filhos). Estes métodos simulam os cruzamentos e mutações que

ocorrem nos indivíduos na natureza e tentam fornecer indivíduos com melhores valores de Fitness, como descrito acima. O método de Mutação, em particular, tem o importante papel de evitar que o algoritmo caia em mínimos locais, permitindo que qualquer indivíduo no espaço de soluções tenha a chance de ser escolhido.

Inicialmente é gerada uma população de indivíduos. Em seguida os operadores genéticos são aplicados nos indivíduos desta população durante um determinado número de vezes, o que é chamado, de forma análoga à evolução natural, de Geração.

Ao final de cada Geração, os indivíduos são selecionados para comporem a próxima geração. Existem Alguns métodos de seleção, sendo que o mais conhecido e que é utilizado neste trabalho é o chamado de Roullete Reell, visto na figura 1, o qual simula uma roleta de jogo. Entretanto sempre os indivíduos com maior Fitness terão maior probabilidade de serem selecionados.

Indivíduo S_i	Aptidão $f(S_i)$	Aptidão Relativa
S_1 10110	2.23	0.14
S_2 11000	7.27	0.47
S_3 11110	1.05	0.07
S_4 01001	3.35	0.21
S_5 00110	1.69	0.11

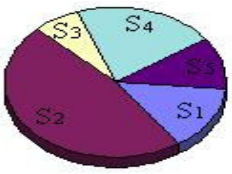


Fig 1: exemplo de método de seleção Roullete Rhell

Ao final de várias gerações a população estará composta por indivíduos com os melhores valores de Fitness ou bem próximo dos melhores. Então se seleciona o melhor.

A figura 2. mostra um exemplo de Algoritmo Genético para uma melhor compreensão do que foi acima descrito:

```
inicializar população P;
repita
    selecione uma subpopulação P'; //pais da próxima geração
    para i ← 1 até nr_cruzamento faça
        escolha S1, S2 ∈ P', aleatoriamente;
        filho ← cruzamento(S1, S2);
        se f(S1) ≥ f(S2) então Saux ← S1;
        senão Saux ← S2;
        se f(Saux) ≥ f(filho) então
            filho substitui Saux em P;
        fim_se;
    fim_para;
    para i ← 1 até nr_mutações faça
        selecione um cromossomo Sj em P;
        Sj ← mutação(Sj);
    fim_para;
até que critério_parada seja satisfeito;
```

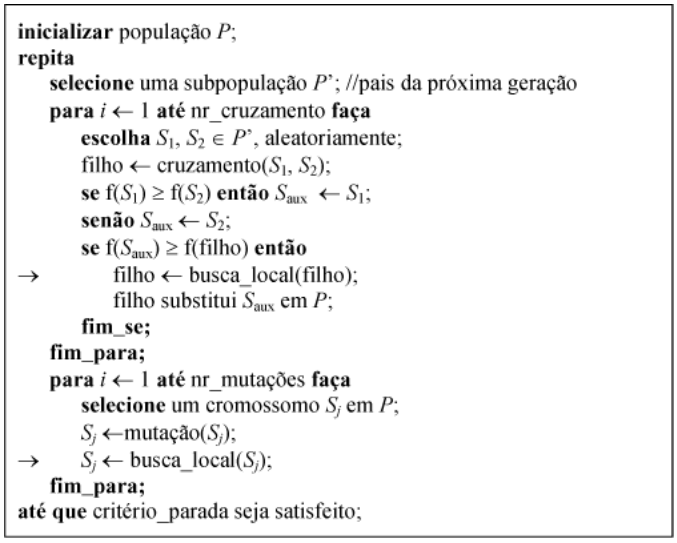
Fig 2. Exemplo de Algoritmo Genético

Os Algoritmos Meméticos são similares em seu conceito aos Algoritmos Genéticos, a diferença é que eles aplicam, além dos operadores genéticos de cruzamento e mutação, métodos de busca local nos indivíduos. Na prática a Busca Local permite que o Algoritmo Memético chegue a soluções mais refinadas do que os Algoritmos Genéticos. Isto é particularmente importante em espaços de busca complexos.

A busca local significa procurar indivíduos (soluções) na vizinhança de um indivíduo inicial, que possuam melhores valores de Fitness ou aptidão. Diversos Algoritmos de Busca Local são empregados. Entre os mais utilizados estão Busca Tabu e Simulated Annealing.

Esta idéia da busca local representa o conceito de "Evolução Cultural" onde um indivíduo, ou seja, uma solução do problema pode a princípio, não ser bom, mas durante o contato e aprendizado com outros indivíduos, aumenta o seu grau de aptidão, ou sua função de Fitness.

A figura 3. mostra um exemplo de Algoritmo Memético para uma melhor compreensão do que foi acima descrito:



```
inicializar população P;
repita
    selecione uma subpopulação P'; //pais da próxima geração
    para i ← 1 até nr_cruzamento faça
        escolha S1, S2 ∈ P', aleatoriamente;
        filho ← cruzamento(S1, S2);
        se f(S1) ≥ f(S2) então Saux ← S1;
        senão Saux ← S2;
        se f(Saux) ≥ f(filho) então
→           filho ← busca_local(filho);
            filho substitui Saux em P;
        fim_se;
    fim_para;
    para i ← 1 até nr_mutações faça
        selecione um cromossomo Sj em P;
        Sj ←mutação(Sj);
→       Sj ← busca_local(Sj);
    fim_para;
até que critério_parada seja satisfeito;
```

Quadro II – Algoritmo memético.

Fig 3. Exemplo de Algoritmo Memético

No contexto do problema que estamos abordando, por exemplo, no subproblema de PAPSP podemos estabelecer o seguinte:

- **Indivíduos (soluções do problema):**

  Podem ser representados como arrays onde suas ***s*** posições representam a alocação do nozzles nos feeders e a seqüência de montagem dos componentes na PCB;

- **Função de Fitness:**

  Neste caso a função de Fitness deve receber como parâmetro de entrada um indivíduo (cromossomo) como o descrito acima e retornar o tempo total gasto para montar a PCB;

- **Operadores Genéticos:**

  Os operadores genéticos de cruzamento e mutação devem ter o cuidado de evitar que os cromossomos filhos resultantes (soluções) tenham componentes ou slots repetidos.